Universidad Politécnica de Madrid

Facultad de Informática

Trabajo Fin de Grado

Prototipo de e-store para un ecosistema y un framework de provisión de servicios

Autor: Francisco de la Vega García

Tutor: Francisco Javier Soriano Camino

Resumen

El trabajo realizado se encuentra enmarcado dentro del proyecto de I+D+I del 7º programa marco de la Comisión Europea *Fi-WARE: The future Internet core platform* que forma parte de la iniciativa Future Internet PPP. En concreto, se ha desarrollado la especificación de un *Generic Enabler* con funcionalidad de tienda virtual que de soporte a la publicación y adquisición o subscripción de aplicaciones y servicios dentro del denominado *Business Framework Ecosystem* (BFE), además de una implementación de referencia de este *Generic Enabler* (GE) que ha sido utilizada para la realización de una prueba de concepto con el objetivo de comprobar la adecuación del comportamiento de la especificación dentro del BFE.

La primera tarea realizada ha consistido en un estudio de otras *stores* (o tiendas digitales) existentes, mirando aspectos tales como la funcionalidad proporcionada, la información mostrada de los distintos productos ofrecidos o la organización de la interfaz de usuario y la metáfora visual. Este estudio ha tenido como objetivo establecer un punto de partida desde el que empezar a analizar las distintas funcionalidades que deberá proveer el sistema.Utilizando como base el estudio anterior y las necesidades concretas de la plataforma Fi-WARE se pasó a la educción de los requisitos generales del sistema en los cuales se especifica a grandes rasgos la funcionalidad que debe proveer esta tienda digital así como algunos aspectos concretos de la experiencia de usuario.

Una vez definida la funcionalidad de la store se ha abordado el diseño del sistema. Para realizar este diseño se ha trabajado en dos tareas principales: La primera de estas tareas ha consistido en realizar el diseño de la arquitectura del *Store GE*, en el que se especifican todos los módulos que debe contener el sistema para poder satisfacer los requisitos, así como las distintas conexiones del *Store GE* con otros componentes del proyecto Fi-Ware y de sus interrelaciones con el resto de componentes de dicho proyecto. Esto ofrece una visión global de la ubicación del *Store GE* dentro de la arquitectura general del proyecto Fi-Ware. La segunda tarea ha consistido en el desarrollo de la especificación abierta (*Open specification*) del *Store GE*. Esta tarea es probablemente la más relevante de cara a cumplir con los objetivos del proyecto Fi-Ware, ya que Fi-Ware se propone como objetivo principal proporcionar las especificaciones de una plataforma tecnológica abierta para la Internet del futuro, formada por un conjunto de componentes (denominados Generic Enablers), entre los que se encuentra el *Store GE*. En este documento ha quedado descrito con todo detalle en que consiste el *Store GE* y cuáles son sus APIs, sobre las que se construirán las aplicaciones de la futura Internet basadas en Fi-Ware, de manera que sea posible que cualquier empresa pueda realizar una implementación diferente a la que se está desarrollando en este proyecto (si bién ésta será su implementación de referencia). Para esta *Open specification* se han desarrollado un modelo de gestión de usuarios y roles, un modelo de datos, diagramas de interacción que definen todas las posibles comunicaciones de la store con otros Generic Enablers

del proyecto Fi-Ware, la definición del ciclo de vida de una oferta y las APIs REST del *Store GE*, incluyendo el contenido de las peticiones y los tipos MIME soportados.

En este punto se pudo comenzar a trabajar en la implementación de referencia del *Store GE*. La primera tarea ha consistido en realizar la integración con el *Marketplace GE*, otro de los Generic Enablers del proyecto Fi-Ware, para ello se definieron unos requisitos específicos y se realizó un diseño de bajo nivel de este módulo seguido de la propia implementación y un conjunto exhaustivo de pruebas unitarias para comprobar su correcto funcionamiento. A continuación se pasó a realizar la integración con el *Repository GE* siguiendo los mismos pasos que con la integración con el *Marketplace GE*.

La siguiente tarea realizada ha consistido en la realización de los módulos necesarios para permitir crear nuevas ofertas en la implementación de referencia de *Store GE* incluyendo nuevamente una fase de educción de requisitos específicos, un diseño de bajo nivel, la propia implementación y una serie de pruebas unitarias.

Una vez implementada la creación de nuevas ofertas, se pasó a la realización de la funcionalidad necesaria para la recuperación y visualización de estas ofertas así como a la realización del soporte necesario para el registro de recursos y para la vinculación de estos a determinadas ofertas, siguiendo nuevamente la metodología antes mencionada.

Finalmente se ha dado el soporte para la publicación y la adquisición de ofertas. En este caso la adquisición de ofertas se ha realizado tan solo en la parte servidora de la aplicación y no se ha llegado a dar soporte a está funcionalidad en la interfaz Web al no ser necesaria para la realización de la prueba de concepto prevista. No obstante esta funcionalidad será implementada junto con otras funcionalidades como el soporte de características sociales, ya fuera del ámbito de este Trabajo de fin de grado.

Como paso previo a la realización de la prueba de concepto se ha trabajado en la plataforma *Wirecloud*, que es una implementación de referencia del denominado *Application Mashup GE*, modificando su funcionalidad para integrarla con la API de compras realizada dentro de la implementación de referencia del *Store GE*.

La última tarea realizada para este Trabajo de fin de grado ha consistido por fin en la realización de la prueba de concepto del *Store GE* integrando su implementación de referencia con las del resto de *Generic Enablers*, lo cual ha permitido comprobar así el fucionamiento de la arquitectura y modelo propuestos.

Índice general

Índice de figuras

Capítulo 1

Introducción

1.1. Contexto del proyecto

1.1.1. El proyecto Fi-WARE

El proyecto *FI-WARE: The future Internet core platform* [28] es un proyecto de I+D+I perteneciente al séptimo programa marco de la Comisión Europea. Este proyecto se propone crear la plataforma núcleo de la Internet del futuro y tiene como objetivo incrementar la competitividad global de la economía Europea basada en tecnología y comunicaciones introduciendo, entre sus principales componentes, una innovadora infraestructura para la creación y distribución de servicios digitales versatiles, con un coste eficiente y ofreciendo una alta calidad de servicio y garantías de seguridad. De esta manera, pretende proveer una base sólida al Internet del futuro estimulando y cultivando un ecosistema sostenible para que proveedores de servicios innovadores distribuyan nuevas aplicaciones y soluciones que satisfagan los requisitos de areas de uso establecidas y emergentes.

Se pretende que FI-WARE sea un sistema abierto, basado en elementos genéricos (que a partir de este punto se denominarán *Generic Enabler*) que ofrezcan funciones reusables y compartidas, útiles en múltiples áreas de uso a través de varios sectores. Es importante indicar que no todas las funciones que sean comunes a varias aplicaciones en un área de uso concreta pueden considerarse *Generic Enablers*, ya que es la habilidad de trabajar en varias áreas de uso lo que distingue a un *Generic Enabler* de lo que se ha denominado en el proyecto *Domain-specific Common Enablers* (o *Specific Enablers*), que son *enablers* comunes a múltiples aplicaciones pero todos ellos específicos de áreas de uso muy limitadas. A modo de ejemplo, un componente de procesamiento de eventos complejos (CEP), un sistema de gestión de identidades (idM) o la propia eStore se consideran *Generic Enablers*. A pesar de que no todos los elementos que puedan ser definidos como *Generic Enablers* serán implementados en el proyecto FI-WARE, se espera que todo lo implementado pueda ser considerado un *Generic Enabler*.

Los objetivos clave del proyecto FI-WARE son la identificación y especificación de los distintos *Generic Enablers*, junto con el desarrollo y demostración de implementaciones de referencia de los *Generic Enablers* identificados. La arquitectura de FI-WARE comprende tanto la especificación de los *Generic Enablers* como las relaciones entre ellos y sus propiedades.

La plataforma que deberá proveer FI-WARE está compuesta por *Generic Enablers* enmarcados dentro de los siguientes capítulos técnicos:

- **Cloud Hosting:** Proveerá recursos de computación, almacenamiento y red.
- **Data/Context management:** Proveerá facilidades de acceso, procesamiento y análisis de volúmenes masivos de información, tranformándolo en conocimiento útil para las aplicaciones.
- **Application/Services ecosystem and Delivery Framework:** Proveerá la infraestructura para crear, publicar, gestionar, y consumir servicios del internet del futuro a través de su ciclo de vida teniendo en cuenta todos los aspectos tecnológicos y de negocio.
- **Internet of Things (IoT) Services Enablement:** Proveerá del puente entre los servicios de la internet del futuro y los dispositivos ubicuos y heterogéneos que forman parte de la *Internet of Things* (IoT).
- **Interface to Networks and Devices (I2ND):** Proveerá de interaces abiertas a redes y dispositivos para satisfacer las necesidades de los servicios ofrecidos a través de la plataforma.
- **Security:** Proveera de los mecanismos necesarios para asegurar que la adquisición y uso de los servicios ofrecidos en la platarforma se realizan de acuerdo a los requisitos de seguridad y privacidad.

La especificación de los distintos *Generic Enablers* se hará mediante las denominadas *Open Specifications* que contendrán toda la información necesaria para poder desarrollar productos que puedan funcionar como implementaciones alternativas del *Generic Enabler* desarrollado en FI-WARE e incluso llegar a reemplazarlo dentro de alguna instancia concreta de FI-WARE. La *Open Specification* de un *Generic enabler* contendrá información como:

- Descripción del ámbito, comportamiento exhibido y uso previsto del *Generic Enabler*.
- Terminología, definiciones y abreviaciones para clarificar los conceptos de la especificación.
- Comportamiento y especificación de las operaciones asociadas a las APIs que el *Generic Enabler* debe exportar. Estas APIs deben estar definidas como una interfaz RESTful.

- Descripción de los protocolos que dan soporte a la interoperabilidad con otros *Generic Enablers* o productos de terceras partes.

- Descripción de características no funcionales.

Finalmente, FI-WARE se enmarca dentro de la *Future Internet Public-Private Partnertship* (FI-PPP) [11], en la cual coexiste con un conjunto de proyectos de casos de uso (*Use Case Projects*) que tienen planeados entre sus objetivos usar los *Generic Enablers* de FI-WARE y el TestBed con sus implementaciones de referencia en sus pruebas de concepto y demandar de FI-WARE las funcionalidades que les son requeridas.

1.1.2. El paquete 3: Applications/Services Ecosystem

El denominado *Applications/Services Ecosystem and Delivery Framework* contendrá una serie de *Generic Enablers* que darán soporte a la gestión de servicios en un marco de negocio a través del ciclo de vida completo de los mismos, desde la creación y composición de servicios hasta la tarificación y reparto de beneficios.

A continuación se describen los principales *Generic enablers* que forman parte de este ecosistema de servicios.

Marketplace GE

El *Marketplace GE* [23] proveé de la funcionalidad necesaria a los proveedores de servicios para publicar sus ofertas y a los consumidores para buscar y comparar las mismas con el objetivo de permitirles seleccionar una implementación concreta del servicio. El *Marketplace GE* permite ofrecer ofertas de diferentes *Store GE* y de diferentes proveedores. El proceso de compra de estas ofertas se realizará en el *Store GE*.

Repository GE

El *Repository GE* [32] es el lugar usado para almacenar los modelos de servicio, especialmente descripciones *Linked USDL* además de otros modelos necesitados por los distintos componentes del *Delivery Framework* como descripciones técnicas. El uso del *Repository GE* para el almacenamiento de estos modelos de servicio es obligado si se quiere registrar estos en el *Marketplace GE* o hacer uso de otras herramientas que requieran la interoperabilidad de *Generic Enablers* Las descripciones publicadas en el *Repository GE* podrán ser utilizadas por cualquier componente de la plataforma FI-WARE teniendo en cuenta las restricciones de privacidad y autorización impuestas por los modelos de negocio.

Registry GE

El *Registry GE* [31] funciona como un servicio de directorio universal usado para el mantenimiento, la administración, el despliege y el acceso a servicios. El *Registry GE* tendrá información de instancias arrancadas de servicios como el *Endpoint*, además de información para crear instancias de estos servicios. Este *Generic Enabler* será usado principalmente por otros *Generic Enablers* como una base de datos común de la que obtener opciones de configuración y propiedades de tiempo de ejecución.

Application Mashup GE

El *Application Mashup GE* [1] permite la composición de pequeñas aplicaciones Web basadas en JavaScript [17] llamadas *widgets* para crear de manera visual aplicaciones más complejas denominadas *Mashups* que incrementan el valor que estos *Widgets* tendrían por separado.

Mediator GE

El *Mediator GE* [24] es básicamente un *Middleware* capaz de proveer interoperabilidad entre los distintos protocolos de comunicación y modelos de datos. Por ejemplo es capaz de convertir los mensajes ASCII de protocolos antiguos como FTP a un mensaje XML para un servicio Web.

1.1.3. Funcionamiento anterior al Store GE

En esta sección se describe el funcionamiento real de las distintas implementaciones de los *Generic Enablers* que forman parte del *Applications/Services Ecosystem* antes de la introducción a esta arquitectura del *Store GE*. Los *Generic Enablers* usados en el proceso de creación y adquisición de ofertas son el *Application Mashup GE* con su implementación de referencia la plataforma *Wirecloud* [26] y los *Generic Enablers Marketplace GE* y *Repository GE* con sus implementaciones de referencia creadas por SAP.

La primera consideración que hay que tener en cuenta es que la plataforma *Wirecloud* solo trabaja con *Widgets*, *Mashups* y *Operators* por lo que estos serán los únicos tres tipos de servicio que es posible ofrecer de manera que se genere un funcionamiento real de la arquitectura descrita.

Por otra parte al no existir un *Store GE* este debe ser simulado en el *Marketplace GE* ya que requiere tener instancias del *Store GE* registradas que serán las que tengan asociadas las ofertas. Para ello se registra un *Store GE* falso con un nombre arbitrario y con su *Endpoint* apuntando al de la instancia del *Repository GE* donde se almacenarán las descripciones de las ofertas en formato *Linked USDL* [19].

Para poder acceder al código de los distintos elementos de la plataforma *Wirecloud*, a la hora de adquirir alguno de ellos, es necesario añadir un campo al documento *Linked USDL* [19] que contenga una URL desde la que sea posible descargar la descripción técnica del elemento propia de la plataforma *Wirecloud*. Dentro de esta descripción técnica existirán enlaces que permitirán la descarga del código, imágen y demás información asociada al elemento.

A la hora de realizar una adquisición, la plataforma *Wirecloud* hace uso del servicio de búsquedas proporcionado por el *Marketplace GE* obteniendo una lista con las URLs apuntando a las descripciones *Linked USDL* [19] contenidas en el *Repository GE* de las distintas ofertas. La plataforma *Wirecloud* usará estas URLs para acceder a las descripciones de las ofertas y visualizar la información de las mismas. Cuando se desea adquirir una de estas ofertas la plataforma *Wirecloud* simplemente descarga la descripción técnica del elemento correspondiente y la usa para cargarlo.

Finalmente, a la hora de publicar una oferta si esta se trata de un *Widget* o un *Operator* este registro deberá hacerse de manera manual usando alguna herramienta que pueda ser usada como cliente REST para almacenar la descripción *Linked USDL* en la instancia del *Repository GE* y para registrar la oferta en la instancia del *Marketplace GE*. En el caso de que se trate de un *Mashup* compuesto en la plataforma *Wirecloud* este proceso será similar pero estará automatizado.

1.2. Motivación del proyecto

El projecto FI-WARE pretende proveer de una infraestructura para la creación y distribución de servicios digitales. Esta distribución no siempre tiene por qué ser gratuita, por lo que que se requiere de un modelo de negocio que permita a los distintos proveedores de servicios trabajar utilizando la plataforma FI-WARE. Esta necesidad de un modelo de negocio fuerza a la existencia de un *Generic Enabler* capaz de gestionar la compra y venta de aplicaciones y servicios.

Por otra parte las distintas soluciones comerciales existentes no son lo suficientemente generales, sino que la mayoría de ellas se basan tan solo en la venta de applicaciones de algún tipo y otras, en menor medida, ofrecen algún tipo de servicio digital. No cubren por tanto las necesidades del proyecto FI-WARE, que especifican un modelo general de ofertas que permita ofrecer servicios, aplicaciones o incluso una combinación de ambos en una sola oferta. Adicionalmente, las soluciones comerciales disponibles no ofrecen un conjunto de especificaciones abiertas como requiere una plataforma tipo FI-WARE

Finalmente también hay que tener en cuenta la existencia del *Applications/Services Ecosystem* en el que los distintos servicios son descritos mediante documentos *Linked USDL* [19] y en el que existen *Generic Enablers* con los que esta tienda digital debe estar integrada, lo que imposibilita el uso de alguna solución ya existente.

Por todo lo expuesto anteriormente se decidió que era necesario tener un *Generic enabler* con funcionalidad de tienda digital capaz de satisfacer las necesidades de los proveedores de servicios del proyecto FI-WARE además de tener una implementación de referencia del mismo.

1.3. Objetivos del proyecto

El objetivo principal de este proyecto es realizar la *Open Specification* de un *Generic Enabler* (en adelante denominado *Store GE*) con funcionalidad de tienda digital de servicios y aplicaciones capaz de manejar un módelo de negocio adecuado para su utilización no solo por usuarios como ocurre con la mayoria de *Stores* existentes sino que sea capaz de gestionar su utilización por compañias u organizaciones permitiendo además un modelo flexible de ofertas que no limite a la venta de un solo servicio u aplicación.

Este *Store GE* que se está definiendo debe estar perfectamente integrado con el resto de *Generic enablers* de la plataforma FI-WARE, especialmente con los pertenecientes al *Work Package 3: Applications/Services Ecosystem and Delivery Framework* ya que será dentro de ese capítulo donde estará situada esta tienda digital. Por tanto este *Store GE* debe permitir ser registrado en una instancia del *Marketplace GE* donde publicará sus ofertas y debe trabajar con ofertas de servicios descritas en *Linked USDL* [19] almacenadas en instancias del *Repository GE*.

El objetivo del *Store GE* dentro de la plataforma FI-WARE es proveer de la lógica necesaria al *Applications/Services Ecosystem* capaz de implementar un modelo de negocio basado en la compra y venta de servicios y aplicaciones que permita realizar una gestión de las ofertas inexistente en el actual *Test bed*.

El siguiente objetivo de este proyecto es la realización de una primera implementación de referencia básica que permita realizar una prueba de concepto de los diseños incluidos en la *Open Specification* del *Store GE* y comprobar así la correcta integración de este con los distintos *Generic Enablers* de la plataforma FI-WARE además de comprobar de manera básica que el diseño realizado satisface las necesidades específicas del *Applications/Services Ecosystem* para poder así replantear algunos conceptos de la *Open Specification* en caso de ser necesario.

Finalmente, debe tenerse en cuenta la existencia del Applications/Services Ecosystem, en el que existen Generic Enablers con los que esta tienda digital debe estar integrada. Por tanto, la solución que se plantee debe considerar la existencia del resto de GEs del ecosistema de servicios y garantizar su completa integración.

1.4. Estructura del documento

El presente documento esta dividido en varios capítulos que se centran en aspectos diferentes dentro de mi trabajo de fin de grado. El capítulo actual, *Introducción*, contiene información relativa al contexto en el que se ha desarrollado el proyecto además de una explicación de la motivación que ha llevado a la necesidad de desarrollar el *Store GE* y los objetivos principales de este desarrollo.

El segundo capítulo, *Estado del arte*, contiene por un lado información relativa al estado de otras tiendas dígitales ya existentes y por otra parte una breve descripción de las principales tecnologías utilizadas para el desarrollo de la implementación de referencia del *Store GE*.

El tercer capítulo, *Desarrollo*, contiene el diseño del *Store GE* haciendo incapié en su *Open Specification* e incluyendo una descripción de en que consiste la implementación de referencia realizada.

Finalmente el cuarto capítulo, *Conclusiones*, contiene en primer lugar los resultados obtenidos de la realización de la prueba de concepto del *Store GE*, en siguiente lugar contiene las conclusiones personales obtenidas tras la realización de este Trabajo de fin de grado y por último contiene una descripción de las lineas futuras que se seguirán en el *Store GE* una vez terminado este Trabajo de fin de grado.

Capítulo 2

Estado del arte

2.1. Pricipales e-Stores

En esta sección se describen las principales tiendas electrónicas existenes en el mercado. Cada una de las cuales ofrece una aproximación diferente al considerar un tipo diferente de producto y por ello han sido utilizadas como punto de partida para obtener algunos requisitos tanto de funcionalidades que deberia proporcionar la store que se ha diseñado como de experiencia de usuario e interfaz gráfica.

2.1.1. Google Play

Google play [13] es la principal *store* para el sistema operativo Android. En ella se pueden encontrar tanto apps como libros en formato digital y películas en alta definición. Esta *store* es accesible tanto desde la aplicación contenida en los smartphone Android como a través de la web. La siguiente descripción se centra en la versión web ya que es la que guarda mayor relación con la propuesta de eStore de este trabajo.

La página de inicio esta dividida en varias áreas. En la parte superior se encuentra la barra de búsqueda, junto con un botón desplegable que permite acceder directamente a comprar apps, libros dispositivos o alquilar películas. Esta sección de la página sirve para la navegación y permanece fija en toda la web.

Debajo se encuentra el menú principal de la página con enlaces a las páginas de apps, libros, películas y dispositivos. Es importante resaltar que estos enlaces conducen a la mismas páginas que el botón desplegable. En siguiente lugar se encuentra el área de recomendaciones personalizadas en la que se muestran apps, libros y películas dependiendo del usuario y su perfil de compras anteriores. Por último esta página contiene una sección dedicada a cada tipo de producto que incluye el top 5 de ese producto, así como una serie de productos destacados. Estas secciones contienen enlaces para ver el top al completo o todos los productos de ese tipo.

La página de apps se divide en dos secciones principales: la lista de apps con una pestaña que permite cambiar entre la selección de apps y la selección de apps para tablet. Éstas contienen varias subsecciones tales como recomendaciones personalizadas, aplicaciones populares y juegos superventas. Para cada una de estas apps se muestra su imagen, su nombre su proveedor, su valoración, y su precio incluido en el botón para adquirirla. La segunda sección permite seleccionar mediante una pestaña entre los Top de aplicaciones tales como top ventas, top gratis, top en ingresos, top ventas nuevo y top gratis nuevo, y una selección por categorías divididas en aplicaciones y juegos.

La página con los detalles de la app contiene en la parte superior la información mostrada en la lista de apps incluyendo el botón de compra, junto a esta información se encuentra una imagen de mayor tamaño y detalle asociada a la app. Debajo se encuentra la información detallada de la app contenida en varias pestañas. La primera contiene la información general incluyendo la descripción, capturas de pantalla y vídeos, valoración con el número de usuarios que han votado un cierto número de estrellas y la valoración media, por ultimo contiene opiniones de los usuarios. La segunda pestaña esta dedicada a las opiniones de los usuarios y contiene las valoraciones y comentarios contenidos en la pestaña anterior. La tercera pestaña contiene la novedades que incluye esta versión de la app respecto de la versión anterior. La última pestaña contiene los permisos que se tendrán que otorgar a la app para que esta funcione correctamente. A parte de la información de la app, esta página contiene una sección de sugerencias al cliente en la que se indican otras apps de mismo proveedor, otras consultadas por los usuarios que consultaron esa aplicación y otras instaladas por los usuarios que la compraron.

Todos los productos de esta store requieren un pago único o son gratuitos, excepto las películas, en las que el pago se corresponde con un alquiler de la misma y por tanto tiene caducidad. Esto es de especial relevancia para el proyecto, ya que en el caso del *Store GE* deberá considerarse además la modalidad de "pago por uso" asociada a la oferta de servicios en lugar de apps.

2.1.2. AWS Marketplace (Amazon web services)

Amazon web services (AWS) es una colección de servicios web que, en conjunto, forman una plataforma de cloud computing ofrecida a través de la web por amazon.com, siendo los principales y más conocidos Amazon EC2 (*Elastic cloud computing*) y Amazon S3 (*Simple Storage Service*). AWS Marketplace [2] es una *store* proporcionada por Amazon en la que se ofrece software servidor que puede ser rápidamente desplegado en su sistema de cloud englobando gran cantidad de aplicaciones como software empresarial, servidores de aplicaciones o comercio electrónico.

La parte superior de la web contiene la zona de navegación, compuesta por una barra de búsqueda, un menú desplegable en el que se pueden seleccionar distintas categorías y un enlace para consultar el software ya adquirido, esta zona está disponible en todas las páginas contenidas en la web.

La página principal contiene por una lado varios grupos de productos incluyendo productos destacados, productos populares, nuevos productos y productos para desarrolladores. Además contiene un menú por categorías que se corresponde con el menú desplegable de la zona de navegación.

Cuando se realiza una búsqueda o se selecciona una categoría, aparece el número de productos que satisfacen la búsqueda realizada, así como un listado de los mismos incluyendo una imagen, el nombre del producto, la versión, el proveedor, el sistema operativo y la máquina que requiere, el precio y parte de la descripción. Además, se muestra un menú en el que se indican las categorías a las que pertenecen los productos mostrados y el número de productos en cada categoría. En caso de que la búsqueda no produzca resultados, se muestran los productos destacados.

La página de detalles de un producto contiene en primer lugar la imagen junto a la descripción del mismo. Debajo se encuentra la información detallada del producto incluyendo la valoración del mismo con un enlace para leer los comentarios, la última versión disponible, el sistema operativo que necesita, la máquina en la que debe desplegarse, los servicios de cloud que hay que contratar para usar el producto, y los puntos fuertes. En siguiente lugar se encuentra la descripción completa del producto, detalles de soporte, recursos, política de reembolso y la licencia de uso. Por último, esta página contiene la tarificación detallada del producto con información del precio del software, y del servicio de cloud necesario para su ejecución así como la región en la que se encuentra disponible.

Esta *store* permite una tarificación muy diversa, en la que principalmente los productos requieren un pago por uso que además incluye el pago del servicio de cloud computing. Este precio, además, variará dependiendo de factores tales como el precio del software, de la región y del servicio de cloud contratado.

2.1.3. Vmware solution exchange

Vmware solution exchange [34] es un *store*, perteneciente a Vmware, en el que se puede adquirir software de distintas categorías como bases de datos, sistemas operativos o aplicaciones para usar sobre los sistemas de virtualización que ofrece esta compañía.

La parte superior de la web contiene la barra de navegación dividida por tipos de software, permitiendo realizar una selección por categorías. Esta barra de nave-

gación está presente a lo largo de todas las páginas de la web.

La página principal contiene dos bloques con listas de productos. En el primero se muestran los productos destacados incluyendo una imagen, su nombre, el proveedor, la valoración, y parte de la descripción. El segundo bloque contiene productos más populares o más recientes dependiendo de la opción que se seleccione.

Al seleccionar una categoría o realizar una búsqueda aparece una lista de productos de los cuales se muestra una imagen, su nombre, el proveedor, la valoración y parte de la descripción. Se puede elegir la ordenación de la lista.

La página con los detalles del producto contiene en primer lugar el nombre del producto, una descripción, las categorías a las que pertenece y el proveedor. A continuación se encuentra la información detallada del producto contenida en pestañas. La primera contiene la información general del producto incluyendo sus puntos destacados y su descripción detallada. La segunda pestaña contiene especificaciones técnicas. La tercera pestaña contiene información de soporte incluyendo horario, números de teléfono, sitio web, email y lista de soluciones compatibles. La siguiente pestaña contiene una lista de partners del proveedor y la última una serie de recursos. Por último la página contiene la valoración del producto.

2.1.4. Hubspot

Hubspot es un software que integra diversas funcionalidades de *marketing* para empresas que operan en la web, tales como optimización para maximizar las oportunidades de que el sitio web sea encontrado o funcionalidades de red social que permitan crear un público. *Hubspot marketplace* [14] ofrece una serie de apps y servicios que pueden ser integrados al software para adaptarlo a las necesidades de la empresa. Está formado por dos marketplaces separados: el *app Marketplace* y el *service Marketplace.*

La página principal está formada por dos zonas que permiten elegir entre el *app marketplace* y el *service marketplace* con algunos de los productos que cada uno de ellos ofrecen.

La página principal del *app marketplace* [15] está compuesta por una zona de navegación en la que se encuentran enlaces para volver a la página principal, para ver las app mejor valoradas, las más nuevas o las más populares. Además contiene una zona de selección por categorías, y una zona con distintos recursos de la web tales como dar una idea, página para desarrolladores, encontrar soporte, notificar un abuso o cambiar al *services marketplace* [16]. Esta zona permanece fija en todas las páginas de la web.

La zona central de la página principal contiene algunas app agrupadas en las categorías de aplicaciones destacadas, mejor valoradas y más nuevas.

Al realizar una búsqueda o seleccionar una categoría, aparece un listado de apps para las que se ofrece una imagen, el proveedor, y la valoración de la app. Además, al pasar el ratón sobre la imagen aparece la descripción de la misma.

La página de detalles de una app contiene en primer lugar el nombre, la valoración, y el nombre del proveedor, así como un enlace para escribir un comentario. A continuación se encuentra una imagen detallada de la app, una descripción, el precio, soporte, el número de veces que se ha instalado la app, y una serie de TAGs. Por último se muestra una serie de capturas de pantalla y los comentarios de los usuarios que incluyen una valoración, el nombre del usuario y la fecha del comentario.

Esta store contiene tanto aplicaciones que pueden adquirirse gratuitamente como aplicaciones que requieren un pago único o un pago periódico.

El *service marketplace* [16] pone a disposición de los clientes una serie de proveedores de servicios relacionados con el marketing agrupados en varias categorías, así como algunos productos y ejemplos del trabajo de marketing en la web de varios de los proveedores registrados.

La página principal del *service marketplace* [16] está compuesta por dos áreas principales. En la primera aparece un listado de categorías relacionadas con el marketing y el diseño web, dividido en creación, optimización, promoción, conversión y análisis. Adicionalmente contiene un campo con distintas funcionalidades proporcionadas por la web como búsqueda con Google, marketing, entrenamiento básico en Hubspot, ayuda y cambio al app marketplace. Por último contiene una sección de proveedores en la que se encuentra el enlace para registrase como nuevo proveedor o hacer login como proveedor existente. El segundo área contiene algunas categorías basadas en la funcionalidad del software de Hubspot.

Al seleccionar una categoría, se puede mostrar información de proveedores, productos de Hubspot o ejemplos, seleccionando el botón correspondiente en la parte superior de la página. La lista de proveedores se muestra en la parte central de la página y permite filtrar por categoría, tipo de servicio (B2B/B2C) e industria, además de ordenarlos. De cada proveedor se muestra su nombre, su valoración, una imagen, la ubicación de la organización y el precio de partida de sus servicios. De los productos disponibles para comprar se muestra su nombre, el inicio de su descripción una imagen y el proveedor del producto, al hacer click sobre la imagen se muestra la descripción completa del producto. Por último la lista de ejemplos contiene una imagen y el proveedor del ejemplo, al dar click sobre el ejemplo aparece una pequeña descripción y la posibilidad de contactar con el proveedor.

La página de detalles del proveedor, contiene tres áreas principales. En la primera se muestra la información de proveedor conteniendo el nombre, su descripción, su logo, la ubicación de la empresa y una lista de las industrias en la que se encuentran sus clientes. El segundo área contiene información de los servicios que ofrecen incluyendo una lista de categorías en la que el proveedor tiene algún servicio, e indicando el número de servicios de cada categoría. Al seleccionar una categoría aparece información de los servicios contenidos en la misma, incluyendo el precio de base, una descripción general, servicios que ofrecen, número de clientes junto con las industrias en las que se encuentran, capturas de pantalla y opiniones y valoraciones. El último área contiene un formulario para ponerse en contacto con el proveedor.

2.1.5. Cherokee Market

Cherokee [4] es un servidor web de alto rendimiento de código libre disponible para los principales sistemas linux. El Cherokee market [3] es una store en la que se pueden encontrar distintas aplicaciones web que pueden integrarse con la distribución de Cherokee.

Cherokee market es una *store* muy sencilla compuesta por una única página con dos secciones. En la primera aparece la lista de servicios disponibles seleccionables por categorías, y en la segunda se muestra información del servicio seleccionado incluyendo el nombre, la categoría, el precio, la descripción, y capturas de pantalla.

Los servicios contenidos en esta store son gratuitos o requieren un pago único.

2.2. Tecnologías utilizadas

En esta seccion se describen una serie de tecnologías utilizadas para la realización de la implementación de referencia del *Store GE* incluyendo lenguajes de programación, frameworks, etc.

2.2.1. RDF

El RDF [30] o *Resource Description Framework* es una familia de especificaciones del W3C (*World Wide Web Consortium*) usadas como un método general para describir y modelar información conceptual que se implementa como recursos web. El modelo de datos del RDF se basa en una serie de sentencias usadas para describir recursos con la forma sujeto-predicado-objeto denominadas tripletas. Este modelo de datos conceptualmente es similar a modelos como el entidad-relacion o a digramas de clases, de manera que el sujeto representa la entidad u objeto que se está representando, el predicado indica un atributo o propiedad del mismo y el

objeto representa el valor de esta propiedad.

El RDF [30] se basa en la idea de identificar los recursos usando indentificadores Web o URIs (*Uniform Resource Identifier*). De esta manera un recurso concreto queda inequivocamente identificado en la Web. Es importante resaltar que, a pesar de que algunas de estos URIs puedan ser URLs, que tienen asociado un recurso real con el que es posible interactuar, no todos tienen por qué serlo ya que es posible representar cualquier concepto.

En la figura 2.1 puede verse un ejemplo extraído de la especificación del RDF [30] del W3C, en el que se representa como un grafo la información de una persona llamada Eric Miller, identificada por el URI http://www.w3.org/People/EM/contact#me

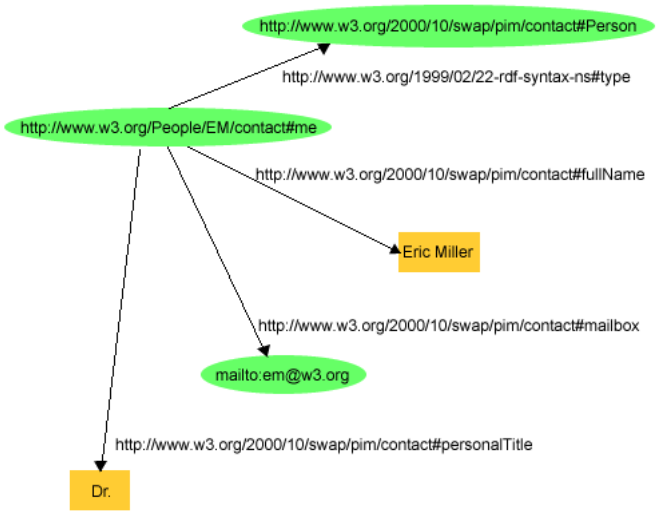

Figura 2.1: Grafo RDF representando a Eric Miller

En la figura 2.1 puede verse como se representa la información utilizando URIs, tanto la entidad principal como grupos o conceptos (ej. *Person* http://www.w3.org/2000/10/swap/pim/contact#Person) y propiedades (ej. *Mailbox* http://www.w3.org/2000/10/swap/pim/contact#mailbox). Por ultimo puede observarse como los valores de estas propiedades pueden ser *Strings* como en "Eric Miller" o "Dr.".

Para representar el RDF [30] existen numerosos formatos de serialización, siendo el primero en aparecer el formato en XML comúnmente llamado RDF, ya que fue introducido por el W3C simultanemente con la especificación del RDF [30]. Un documento RDF serializado en XML esta compuesto básicamente por un nodo raíz etiquetado como rdf:RDF, en el que se incluyen las distintas definiciones de espacios de nombres asociados a los vocabularios que van a ser utilizados dentro del documento. Dentro de este nodo se situarán las descripciones de los recursos, etiquetadas

como rdf:Description, con el atributo rdf:about indicando el URI del recurso. Por último, esta descripción contrendrá las propiedades y los valores de estas. A continuación se muestra un ejemplo de este formato de serialización.

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
             xmlns:exterms="http://www.example.org/terms/">

  <rdf:Description rdf:about="http://www.example.org/index.html">
      <exterms:creation-date>August 16, 1999</exterms:creation-date>
  </rdf:Description>

</rdf:RDF>
```

Además de este formato de serialización, se han definido una serie de formatos de representación en texto plano que tienen un formato más legible como por ejemplo N-Triples [8] en el que simplemente se escriben las tripletas con los URIs completos. O los formatos Turtle [10] y Notation3 [9] (o N3). que son formatos derivados del N-Triples [8] más avanzados, en los que es posible por ejemplo declarar los espacios de nombres o agrupar propiedades y valores de propiedades. A continuación se muestra un ejemplo del formato de serialización Turtle[10].

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix ex: <http://example.org/stuff/1.0/> .

<http://www.w3.org/TR/rdf-syntax-grammar>
  dc:title "RDF/XML Syntax Specification (Revised)" ;
  ex:editor [
    ex:fullname "Dave Beckett";
    ex:homePage <http://purl.org/net/dajobe/>
  ] .
```

A parte de los formatos anteriores, existen algunos formatos concebidos con el objetivo de facilitar el transporte de la información contenida en un RDF en comunicaciones basadas en JSON, como por ejemplo el formato JSON-LD [7] (o *JavaScript Object Notation for Linked Data*) basado en la creación de una stuctura de datos *context* encargada de enlazar las propiedades del recurso contenidas en el documento JSON con los conceptos de una ontología. A continuación se muestra un ejemplo del formato de serialización JSON-LD [7].

```
{
  "@context": {
    "name": "http://xmlns.com/foaf/0.1/name",
    "homepage": {
      "@id": "http://xmlns.com/foaf/0.1/workplaceHomepage",
      "@type": "@id"
    },
    "person": "http://xmlns.com/foaf/0.1/Person"
  },
  "@id": "http://me.markus-lanthaler.com",
  "@type": "person",
  "name": "Markus Lanthaler",
  "homepage": "http://www.tugraz.at/"
}
```

2.2.2. Linked USDL

Linked USDL [19] es una versión del lenguaje de descripción de servicios USDL [33] (*Unified Service Description Languaje*) realizada con el objetivo de promover el uso de este lenguaje de descripción en un entorno web. El USDL [33] entiende un servicio como una transacción económica o social que tiene asociada una información de contexto para la cual es necesario describir el esquema de precios, los terminos y condiciones que deben satisfacer las distintas partes implicadas a la hora de consumir y pagar por el servicio, los distintos interesados en este servicio incluyendo terceras partes e intermediarios y la información técnica asociada con el mismo.

Para representar esta información el USDL [33] permite una descripción unificada de aspectos técnicos, operacionales y de negocio. El USDL [33] permite representar no solo servicios entididos en el ámbito de las tecnologías de la información tales como un servicio SOAP o REST sino que permite representar cualquier tipo de servicio incluyendo servicios manuales o físicos permitiendo una sencilla agregación en el caso de tener un servicio híbrido.

El Linked USDL [19] adapta el USDL [33] a los principios del la web semántica y el *Linked data* creando una serie de ontologías que permiten definir un servicio en un documento RDF [30]. Este vocabulario no define todos los conceptos de USDL sino que hace uso de ontologías ya existentes tales como FOAF, GoodRelations, DCTERMS, MSM, ORG, SKOS etc. permitiendo que un documento Linked USDL [19] sea facilmente extensible y adaptable a las necesidades propias del servicio.

Linked USDL [19] define tres vocabularios diferentes para representar distintos aspectos del servicio. Estos vocabularios son USDL-Core [20], USDL-Pricing [21], USDL-SLA [22]. El USDL-Core define información principal del servicio e incluye información relativa a su caracterización asi como a recursos utilizados, relaciones con otros servicios y composición e interaciones posibles con el servicio. El USDL-

Pricing define un vocabulario sencillo para representar información asociada al precio del servicio y a su modelo de negocio. Por último el USDL-SLA define un vocabulario que permite describir los términos y condiciones tanto funcionales como no funcionales a los que está sujeto el uso del servicio.

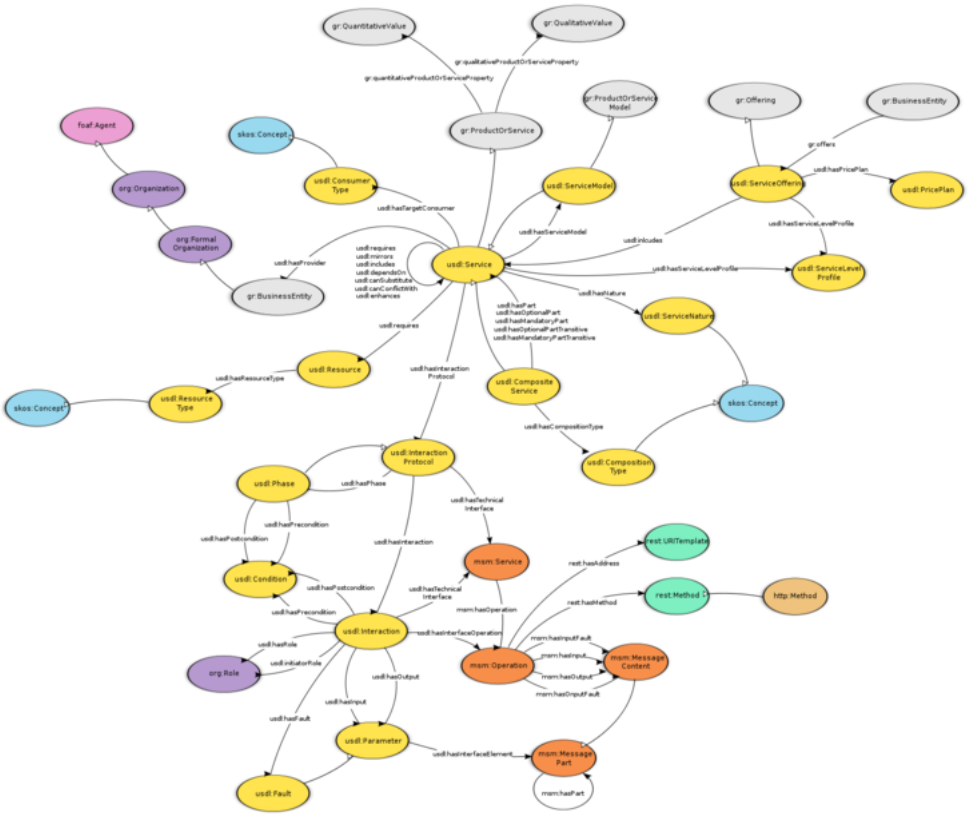

Figura 2.2: Vocabulario USDL-Core

2.2.3. Python

Python [29] es un lenguaje de programación creado a finales de los 80 por Guido van Rossum en el CWI o *Centrum Wiskunde & Informatica* (Centro para las matemáticas y la informática) en los paises bajos, como sucesor del lenguaje de programación ABC. Python [29] es un lenguaje de programación interpretado, con tipado dinámico y multiplataforma, considerado como un lenguaje de programación multiparadigma ya que soporta orientación a objetos, programación imperativa y algo de programación funcional. Python [29] actualmente es administrado por la *Python Software Foundation* bajo una licencia de código abierto denominada *Python Software Foundation License*, compatible con la Licencia pública general de GNU a partir de la versión 2.1.1.

Python [29] es un lenguaje de programación diseñado para ser leído con facilidad agrupando los distintos bloques mediante espacios o tabuladores indistintamente siempre que todos los elementos de un mismo bloque tengan la misma indentación. No obstante se recomienda no mezclarlos. De la misma manera Python [29] utliza palabras reservadas donde otros lenguajes utilizan símbolos, como por ejemplo en el caso de los operadores lógicos. Es importante resaltar tambien que en general un progama escrito en Python [29] será mas corto que sus equivalentes en C/C++ o Java ya que Python [29] contiene tipos de datos complejos nativos del lenguaje, tales como diccionarios, listas y tuplas, lo que permite realizar operaciones complejas en una sola linea. De igual manera, el tipado dinámico mencionado anteriormente hace que no sea necesaria la declaración de variables, sino que éstas se crean y se les asigna un tipo la primera vez que se les asigna un valor, siendo incluso posible que una variable cambie su tipo a lo largo de la ejecución del programa. A continuación se muestra un ejemplo de código Python [29].

```
1 def factorial(x):
2     if x == 0:
3         return 1
4     else:
5         return x * factorial(x - 1)
```

2.2.4. JavaScript

JavaScript [17] es un lenguaje de programación originalmente desarrollado en *Netscape* por Brendan Eich bajo el nombre de Mocha. Paso a llamarse LiveScript como su primer nombre oficial al ser introducido en las versiones beta del *Netscape Navigator 2.0*. Finalmente fue renombrado como JavaScript [17] cuando fue incluido en el navegador *Netscape* en su versión 2.0B3. JavaScript [17] es un lenguaje de programación interpretado normalmente incluido en los navegadores web para crear interfaces de usuario y páginas web dinámicas cuyos principios de diseño fueron extraídos de los lenguajes de programación Self y Scheme. JavaScript [17] es un lenguaje de programación multiparadigma que soporta programación orientada a objetos, funcional e imperativa. Por otra parte JavaScript [17] es un lenguaje basado en prototipos [27], es decir, no existe el concepto de clase y la herencia se realiza mediante el clonado de objetos existentes que funcionan como prototipos. JavaScript utiliza además un sistema de tipos dinámico y débil. Actualmente JavaScript [17] es una marca registrada de *Oracle corporation* y está bajo la licencia para tecnología inventada y desarrollada por *Netscape Communications* y entidades actuales como la *Mozilla Foundation*. A continuación puede verse un ejemplo de código JavaScript [17].

```
var displayClosure = function() {
    var count = 0;
    return function () {
        return ++count;
    };
}
var inc = displayClosure();
inc(); // returns 1
inc(); // returns 2
inc(); // returns 3
```

El uso más común de JavaScript [17] es incluir código dentro de documentos HTML que interactúe con el árbol DOM (*Document Object Model*) de la página para realizar acciones como realizar peticiones al servidor mediante peticiones AJAX, realizar la animación de elementos de la página, validar entradas de formularios en el lado cliente sin necesidad de realizar la petición al servidor o para crear contenido interactivo como juegos, música o video. A continuación puede verse un pequeño ejemplo de código JavaScript [17] incrustrado en un documento HTML.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD␣HTML␣4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
  <head><title>simple page</title></head>
  <body>
    <h1 id="header">This is JavaScript</h1>
    <script type="application/javascript">
      document.body.appendChild(document.createTextNode('Hello World!'));
      var h1 = document.getElementById("header");
      h1 = document.getElementsByTagName("h1")[0];
    </script>
  </body>
</html>
```

2.2.5. JQuery

JQuery [18] es una libreria para JavaScript [17] creada por John Resig que permite interactuar con el árbol DOM de un documento HTML, permitiendo modificar nodos y eventos de manera sencilla, crear animaciones y realizar peticiones AJAX. JQuery [18] es software libre y de código abierto bajo doble licencia: la Licencia MIT y la Licencia Pública General de GNU en su versión 2, permitiendo de esta manera que JQuery [18] pueda ser utilizada tanto en proyectos libres como en proyectos privados.

JQuery [18] se basa en el uso de la función *$()*, la cual recibe como parámetro un selector CSS que permite obtener el nodo o nodos del árbol DOM con los que se quiere trabajar. Una vez obtenidos estos nodos ya es posible aplicar las distintas funciones que provee JQuery [18] o incluso algún efecto gráfico. A continuación

puede verse un ejemplo del uso de JQuery [18].

```
1 $(".activo");
2 $(".activo").removeClass("activo").addClass("inactivo");
3 $(".activo").slideToggle("slow");
```

2.2.6. Django

Django [6] es un *framework* para el desarrollo de aplicaciones Web de código libre escrito en Python [29], principalmente pensado para el desarrollo de aplicaciones Web centradas en su base de datos. Django [6] se basa en un modelo arquitectónico Modelo-Vista-Controlador enfatizando la reusabilidad, la agregación de nuevos componentes y un rápido desarrollo. Actualmente Django [6] se distribuye bajo la Licencia BSD y es mantenido por la *Django Software Foundation*.

Para dar soprte a este modelo arquitectónico Modelo-Vista-Controlador, Django [6] utiliza una serie de *scripts* Python [29] en los que se provee la información o funcionalidades necesarias dependiendo de la aplicación que se esté desarrollando. Para trabajar con los modelos, Django [6] utiliza el fichero models.py en el que se definen las distintas tablas de la base de datos utilizando código Python [29], para ello cada tabla quedará definida con una clase que herede de la clase Model definida en Django [6]. Dentro de esta clase se definirán los distintos atributos usando las clases definidas en Django [6]. Por otra parte Django [6] provee una vista de administración desde la que es posible trabajar con los modelos de la base de datos definidos a través de un navegador Web. A continuación se muestra un ejemplo de un modelo definido en Django[6].

```
1 class Article(models.Model):
2     pub_date = models.DateTimeField()
3     headline = models.CharField(max_length=200)
4     content = models.TextField()
5     reporter = models.ForeignKey(Reporter)
6
7     def __unicode__(self):
8         return self.headline
```

Para trabajar con las vistas, Django [6] utiliza un sistema de templates en los cuales se escribe código HTML en el que además es posible incluir variables (entre doble llave {}) y código similar al Python [29] (escrito entre {%%}) que generán la página Web a mostrar de manera dinámica en tiempo de ejecución. A continuación puede verse un ejemplo de template de Django [6].

```
1  {% extends "base.html" %}
2
3  {% block title %}Articles for {{ year }}{% endblock %}
4
5  {% block content %}
6  <h1>Articles for {{ year }}</h1>
7
8  {% for article in article_list %}
9      <p>{{ article.headline }}</p>
10     <p>By {{ article.reporter.full_name }}</p>
11     <p>Published {{ article.pub_date|date:"F␣j,␣Y" }}</p>
12 {% endfor %}
13 {% endblock %}
```

Finalmente el control en Django [6] se realiza utilizando los ficheros urls.py y views.py. El fichero urls.py contendrá una serie de patrones que asocian determinadas URLs con funciones o clases definidas dentro del fichero views.py de manera que cuando en Django [6] se recive una petición HTTP en una URL se pondrá en ejecución la función correspondiente. Esta función podrá tanto mostrar una vista realizando el mapeo de las variables de un template y devolviendo el código HTML correspondiente, como realizar alguna funcionalidad en el caso de que su URL forme parte de alguna API.

2.2.7. MongoDB

MongoDB [25] es una base de datos no relacional, basada en documentos y de código abierto, desarrollada por 10gen en el año 2007. Al contrario que en las bases de datos relacionales, MongoDB [25] no almacena la información en tablas sino que almacena la información en documentos JSON a los que denomina BSON siguiendo un esquema dinámico, es decir, no todos los documentos tenen porque tener la misma estructura. Actualmente MongoDB se distibuye bajo la licencia AGPL.

MongoDB se organiza en colecciones, las cuales contenen documentos. En el caso de una base de datos relacional, una colección podría considerarse como el equivalente de una tabla o entidad mientras que los distintos documentos podrian considerarse como las entradas de esta tabla. No obstante como se mencionó anteriormente MongoDB [25] utiliza un esquema flexible, por lo que no es necesario que todos los documentos de una colección tengan la misma estructura ni que un mismo campo de dos documentos de una colección tengan el mismo tipo de datos. Esto permite que en un documento sólo aparezcan los campos relevantes y no sea necesario almacenar información inútil.

Las diferencias planteadas anteriomente entre MongoDB [25] y las bases de datos relacionales hacen que sea necesario realizar algunas consideraciones a la hora de diseñar la estructura básica de los documentos de una colección. En una base de datos relacional las relaciones entre dos entidades diferentes se representarán como

una tabla en el caso de una relación *Many-to-Many* o como una *Foreign Key* en el caso de una relación *Many-to-One*. Esto no resulta tan claro en MongoDB, donde es necesario decidir si se crean dos documentos diferentes y se usan referencias, o si simplemente se incluye un documento dentro de otro. Para resover estos problemas, la documentacion de MongoDB [25] recomienda anidar los documentos cuando se tenga una relación *One-to-One* en la que conceptualmente un documento contiene a otro. Además se recomienda usar esta anidacion de documentos cuando se tenga una relación *One-To-Many* en la que los documentos que participan de manera múltiple siempre aparezcan en el contexto del documento que participa con 1 en la relación. Esta anidación de documentos tiene una mayor eficiencia y permite extraer la información de la base de datos en una sola operación. Por otra parte se recomienda referenciar documentos en el caso de que anidar documentos provoque una replicación de información pero sin que exista una mejora sustancial del rendimiento en el acceso a la información.

Con el objetivo de mejorar la eficiencia, MongoDB [25] tiene soporte para crear índices. Estos índices son similares a los que se utilizan en bases de datos relacionales y están compuestos por una estructura de datos que permite localizar documentos rápidamente basándose en los valores almacenados para ciertos campos del mismo. MongoDB [25] define los índices a nivel de colección, pudiendo definirse índices para un único campo o para varios usando índices complejos.

Finalmente es necesario indicar que MongoDB [25] tiene un sistema de *Querys* basado en patrones, el cual se basa en las operaciones *find* y *findOne*, a las cuales se les pasa como parámetro una estructura JSON conteniendo la información que debe tener el documento o documentos que se pretenden obtener.

2.2.8. GIT

GIT [12] es un sistema de gestión del código o SCM (*Source Code Management*) y control de versiones creado por Linus Torvalds para el desarrollo del *Kernel* del sistema operativo Linux. GIT [12] es software libre y se distribuye bajo la Licencia Pública General GNU en su versión 2.

GIT [12] es un sistema de control de versiones distribuido, por lo que existen uno o varios servidores que contienen las distintas versiones de los ficheros y además los clientes tendrán una copia local completa del repositorio. De esta manera se evitan los problemas que pueden producirse en el caso de que exista un fallo con el servidor y permite que se realicen cambios en local, por lo que se evitan las esperas y latencias propias de la red.

A diferencia de otros sistemas de control de versiones GIT, [12] no almacena directamente todas las versiones de los ficheros, sino que realiza instantaneas del es-

tado de los ficheros y después almacena simplemente los cambios entre las distintas versiones de forma encadenada a partir de la última instantanea.

Para terminar, es necesario indicar los distintos estados en los que puede encontrarse un fichero que se está controlando con GIT [12]. Cuando se realiza un cambio en un fichero éste queda como modificado (*Modified*). Estos cambios pueden deshacerse usando un *checkout* o incluirse en el próximo *commit* mediante un *Add*. Si se incluye en el próximo *commit*, el fichero pasa a estar añadido (*Added*). Desde este estado sigue siendo posible quitar un fichero del proximo *commit* y volver a dejarlo como mofificado. Finalmente los cambios añadidos pasan a almacenarse mediante la operación *commit*. Es importante resaltar que este proceso se realiza en local y que para subir estos cambios al servidor es necesario realizar una operación *Push*. De igual manera para obtener los cambios existentes en el servidor es necesario realizar la operación *Pull*. En la Figura 2.3 puede verse un diagrama de este proceso.

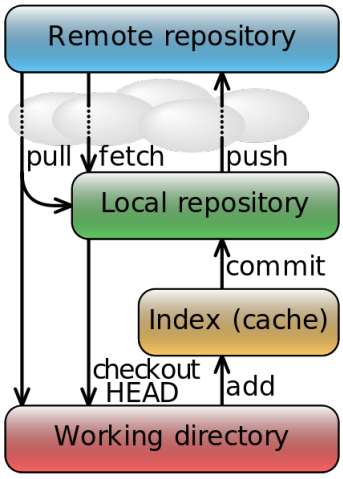

Figura 2.3: Transición de estados en un fichero en GIT

Capítulo 3

Desarrollo

3.1. Diseño de la *Open Specification*

3.1.1. Concepto y requisitos

El *Store GE* es parte del *Application/Services Ecosystem* del proyecto FI-WARE. De manera general el *Store GE* se encarga de proveer gestión de ofertas y ventas: permite publicar nuevas ofertas, gestiona el pago de esas ofertas y ofrece acceso a los distintos servicios adquiridos, facilita la descarga del software en el caso de que la oferta pertenezca a un servicio descargable (ej. Aplicaciones, widgets, etc).

Teniendo en cuenta lo dicho anteriormente se han definido las siguientes características generales para el *Store GE* que suponen a su vez una relación de los requisitos que habrá de cumplir:

- Un *Store GE* se encarga de la gestión de las ofertas de servicios. Para ello permite registrar ofertas publicadas por un *aggregator*, actualiza uno o varios *Marketplace GE* dando de alta la nueva oferta, que queda vinculada al *Store GE*. Además, registra la información de la oferta almacenando el modelo del servicio representado con un documento *Linked USDL* [19] en un *Repository GE*. Finalmente registra la información relativa al tiempo de ejecución del servicio, como las instancias concretas existentes o información de configuranción en el *Registry GE*.

- Ya que no existe ningún *Generic Enabler* en el *Application/Services Ecosystem* encargado del almacenamiento de código, el *Store GE* ofrecerá un *endpoint* desde el que será posible descargar aquellos recursos pertenecientes a un servicio descargable adquirido. Para ello el *Store GE* define una API que el proveedor del servicio podrá implementar y que será utilizada como medio para realizar una descarga indirecta de los recursos asociados a la oferta. Por otra parte el *Store GE* también permite subir estos recursos al propio *Store GE* en el caso en que el proveedor del servicio descargable no disponga de un servidor de aplicaciones.

- El *Store GE* ofrecerá un portal Web que permite visualizar, buscar y adquirir las distintas ofertas independientemente de que el *Marketplace GE* pueda ser usado para buscar y comparar ofertas publicadas en distintos *Store GE*.

- Cuando un comprador haya decidido adquirir una oferta bien a través del portal Web del *Store GE* o bien mediante una búsqueda en el *Marketplace GE*, el *Store GE* gestinorá el pago de la oferta, contactará con el proveedor del servicio para notificar la adquisición y se encargará de que se permita el acceso del comprador al servicio en el caso de un servicio Web o descargará los recursos necesarios como se indicó anteriormente en el caso de un servicio descargable. Adicionalmente el *Store GE* define una API que podrá ser implementada en el caso de que se esté realizando la adquisicion de un servicio descargable a través de un programa cliente para permitir que los recursos del servicio sean automáticamente incluidos en este sistema cliente. Un ejemplo de esta última funcionalidad podría ser la adquisición de widgets a través del *Aplication Mashup GE*.

3.1.2. Modelo de usuarios y roles

Teniendo en cuenta las distintas funcionalidades que proporciona el *Store GE*, es necesario definir un modelo que permita controlar los privilegios y posibles interacciones de los usuarios del *Store GE*. Es necesario indicar que el mantenimiento de la información de los usuarios de la plataforma FI-WARE será realizado por el *Identity management GE*, por lo que el *Store GE* hará uso de la información ofrecida por éste para realizar la gestión de usuarios y roles.

En primer lugar el *Store GE* está basado en el concepto de organización, que será gestionado como un grupo de usuarios. Estas organizaciones permiten trabajar con el *Store GE* de manera que determinadas ofertas puedan ser adquiridas no sólo por el usuario que realiza la compra, sino que pueden quedar disponibles para todos los usuarios dentro de una organización. Del mismo modo que una organización tiene varios usuarios que pertenecen a ella, un mismo usuario puede pertenecer a varias organizaciones distintas, pudiendo tener distintas ofertas adquiridas en cada organización. En el *Store GE* se considera que las ofertas son ofrecidas por una organización en lugar de por un usuario en concreto, por lo que si un usuario desea publicar una oferta propia deberá tener su propia organización. No obtante esto no es incompatible con la pertenencia a otras organizaciones. Como se mencionó anteriormente la gestión de organizaciones será realizada por el *Identity management GE*.

Dependiendo de los privilegios y las distintas interacciones posibles con el *Store GE* se han definido los siguientes roles de usuario. Es importante resaltar que un usuario podrá tener un número arbitrario de roles siempre que tenga al menos un rol.

- **Admin:** Un *Admin* es responsable de la administración del sistema. Esto

incluye la administración de la base de datos así como el registro en el *Store GE* de intancias del *Repository GE* en las que almacenar los modelos de los servicios ofrecidos. Por otra parte también se encarga de registrar la instancia del *Store GE* en las instancias del *Marketplace GE* en los que se pretenda publicar ofertas registradas.

- **Provider:** Tiene la posibilidad de publicar ofertas de servicios que, como se mencionó anteriormente, serán ofrecidas desde la organización en nombre de la cual esté actuando en el momento de la publicación.

- **Customer:** Tiene la posiblidad de adquirir una oferta que podrá pasar o no a formar parte de los servicios adquiridos por alguna de las organizaciones a las que pertenece el usuario.

Es importante resaltar en este punto que no se ha definido ningún rol relativo a las organizaciones ya que eso será tarea del *Identity Management GE*, de la misma forma el *Store GE* no se encarga de gestionar qué rol tiene un usuario para una organización concreta, sino que será el *Identity Management GE* el que se encarge de gestionar esto y simplemente comunicará al *Store GE* qué rol tiene un usuario en un momento concreto y en nombre de que organización está actuando.

3.1.3. Arquitectura

En la Figura 3.1 puede verse la arquitectura del *Store GE* en formato FMC [**?**] dividida en una serie de módulos funcionales así como las relaciones existentes entre el *Store GE* y otros *Generic Enablers* de la plataforma FI-WARE. Es necesario resaltar la importancia del *Application Mashup GE* en este diagrama ya que permite observar como no sólo un usuario es capaz de utilizar el *Store GE* a través de su portal Web, sino que otros *enablers* serán capaces de contactar con la API definida para gestionar y comprar ofertas previamente descubiertas en el *Marketplace GE*. Puede observarse en la Figura 3.1 que el *Store GE* se relacionará ,en primer lugar, con el *Application Mashup GE*, que utilizará el *Store GE* para la adquisición de los distintos componentes Web que utiliza para la composición de *Mashups* así como para realizar la publicación de estos *Mashups* compuestos. Por otra parte el *Store GE* se relacionará tambien con el *Marketplace GE*, que será utilizado para registrar las nuevas ofertas que hayan sido publicadas y con el *Repository GE* para almacenar los documentos *Linked USDL* con la descripción de estas ofertas y sus documentos asociados (WSDL, WADL, etc.). Finalmente, el *Store GE* se relacionará también con el *Mediator GE* y con el *Revenue Share System GE*. El primero será utilizado tanto para acceder a la pasarela de pago mientras se realiza una compra como para contactar con el proveedor del servicio adquirido y comunicarle esta adquisición, mientras que el *Revenue Share System GE* será utilizado para realizar la división del cobro de las ofertas adquiridas entre los distintos *Stakeholders* de éstas.

Como puede verse en la Figura 3.1 el *Store GE* está dividido en módulos, cada uno con una funcionalidad específica, que se relacionan entre ellos o con el exterior

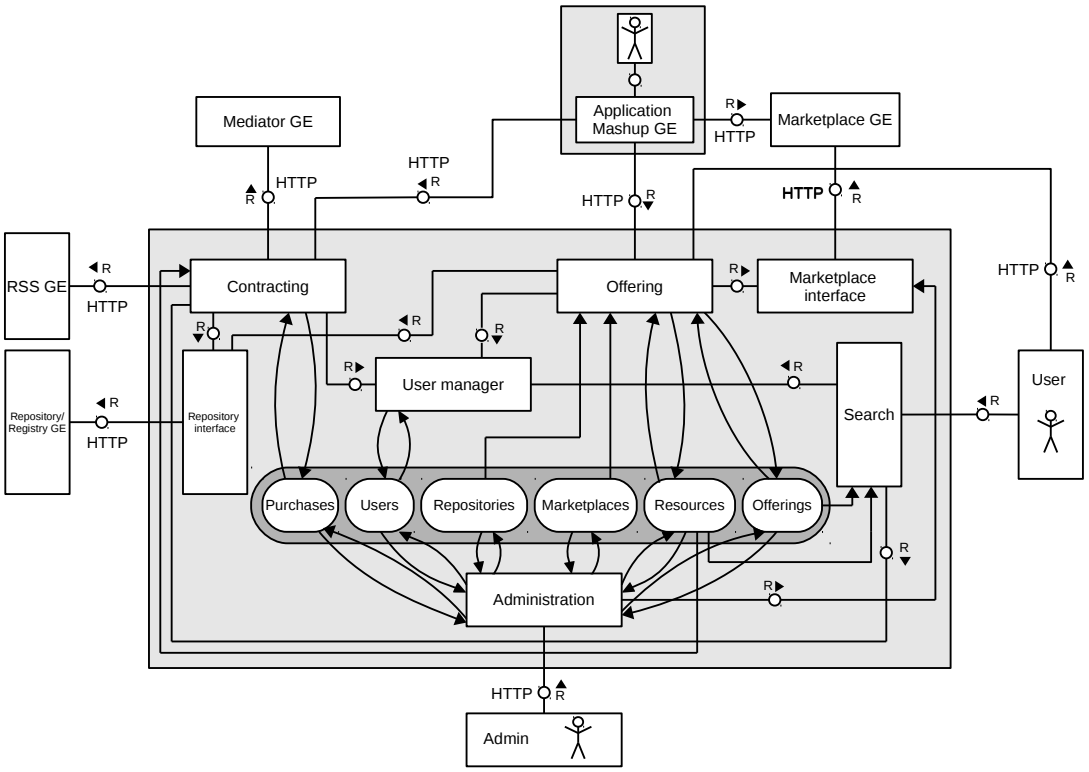

Figura 3.1: Arquitectura del Store GE en formato FMC

del sistema. A continuación se muestra de manera detallada la funcionalidad que deberán proporcionar cada uno de estos módulos.

El módulo *Marketplace Interface* se encarga de las comunicaciones con el *Marketplace GE* para permitir registrar y eliminar la instancia del *Store GE* así como registrar y eliminar ofertas publicadas. Este módulo se comunicará tanto con el módulo *Offering* encargado del registro de ofertas como con el módulo *Administration* que registrará la instancia de *Store GE*.

El módulo *Repository Interface* es el encargado de comunicar con el *Repository GE* tanto para subir como para descargar los documentos *Linked USDL* [19] asocidos a las ofertas que son publicadas. Este módulo se comunicará con el módulo *Offering* del que obtendrá los documentos *Linked USDL* y con el módulo *Contracting* que realizará peticiones para la descarga de estos documentos

El módulo *Administration* es el usado por los usuarios con el rol *Admin* para gestionar el *Store GE* y para realizar el registro de la instancia del *Store GE* en las instancias del *Marketplace GE*. Este módulo lee y escribe en todos los modelos de datos y contacta con el módulo *Marketplace GE* para realizar el registro del *Store GE*

El módulo *Offering* es usado para la gestión de las ofertas propias de los usuarios. Es decir, este módulo realiza la gestión de las ofertas del proveedor, lo que incluye la creación de nuevas ofertas, la publicación de nuevos recursos, la vinculación de recursos a ofertas y la publicación de las ofertas para ponerlas a la venta. Este módulo se encarga también de las ofertas ya adquiridas permitiendo a un comprador acceder a la infomación de éstas así como a sus recursos vinculados. Este módulo lee y escribe de los modelos *Resource* y *Offering* y lee de los modelos *Marketplace* y *Repository* para obtener la información necesaria de las instacias del *Marketplace GE* y *Repository GE* para realizar la publicación de las nuevas ofertas. Además este módulo contacta con el módulo *Marketplace Interface* al que realiza peticiones para gestonar las distintas ofertas que se publican en el *Marketplace GE* y con el *Repository Interface*, el cual utiliza para la subida y descarga de las descripciones *Linked USDL* [19] de las distintas ofertas del usuario. Finalmente, este módulo contacta también con el módulo *User Manager*, que utiliza para la gestión de roles de los usuarios que realizan las peticiones.

El módulo *User Manager* se encarga de la gestión de los usuarios controlando los distintos roles y privilegios para controlar el acceso a las distintas funcionalidades del *Store GE*. Este módulo lee y escribe en el modelo de usuarios y es contactado por todos los módulos que son accesibles desde el exterior del *Store GE*. Es decir, es contactado por los módulos *Offering*, *Search* y *Contracting*.

El módulo *Contracting* es el encargado de la gestión de las subscripciones y compras de las distintas ofertas publicadas en el *Store GE*. Este módulo contactará con las distintas pasarelas de pago, recibirá la confirmación del pago, dará acceso al servicio y le dará al *Revenue Share System GE* la información del pago. Este módulo se encargará además de la renovación de los servicios cuyo modelo de pago sea "pago por subscripción" (Si el servicio permite la subscripción desde fuera del *Store GE*, entonces el consumidor deberá indicar al *Store GE* de manera explícita esta renovación a través de la API de compras). Este módulo se encargará además de permitir a los usuarios ver la información de sus compras. Toda información dinámica (ej. modelo de pago por uso) requerirá el contacto directo del consumidor con el proveedor del servicio. De esta manera el *Store GE* no se hace responsable de la valided de la información ofrecida por el proveedor del servicio. No obstante este módulo permitirá a los usuarios realizar quejas formales acerca de servicios adquiridos. Este módulo lee y escribe del modelo de compras y lee del modelo de recursos. Además es contactado por el módulo *Search* cuando el usuario quiere adquirir un servicio buscado en el *Store GE* y contacta con el módulo *Repository Interface* para obternner las descripciones de la ofertas del *Repository GE*. Este módulo contacta con el *Mediator GE* para la gestión de la compra a través de la pasarela de pagos y para informar a los proveedores de servicios y con el *Revenue Share System GE* con la información de los pagos para realizar su distribución entre los distintos *Stakeholdeers*.

Finalmente, el módulo *Search* se encarga de realizar búsquedas de ofertas pub-

licadas dependiendo de ciertos parámetros y filtros proporcionados por el usuario. Este módulo lee de los modelos *Offering* y *Resource* para poder obtener la información necesaria para realizar la búsquedas y contacta con el módulo *Contracting* para permitir al usuario adquirir una oferta que ha sido buscada.

3.1.4. Modelo de datos

La primera consideración que es necesario realizar acerca del modelo de datos es la división existente del concepto de oferta de un servicio. Para manejar este concepto el *Store GE* trabaja en primer lugar con el modelo *Offering* que representa la entidad asbtracta de una oferta incluyendo información de tarificación, legal o de nivel de servicio. Por otra parte el *Store GE* utiliza también el modelo *Resource* que representa los elementos realmente ofrecidos tales como una aplicación o acceso a una API concreta.

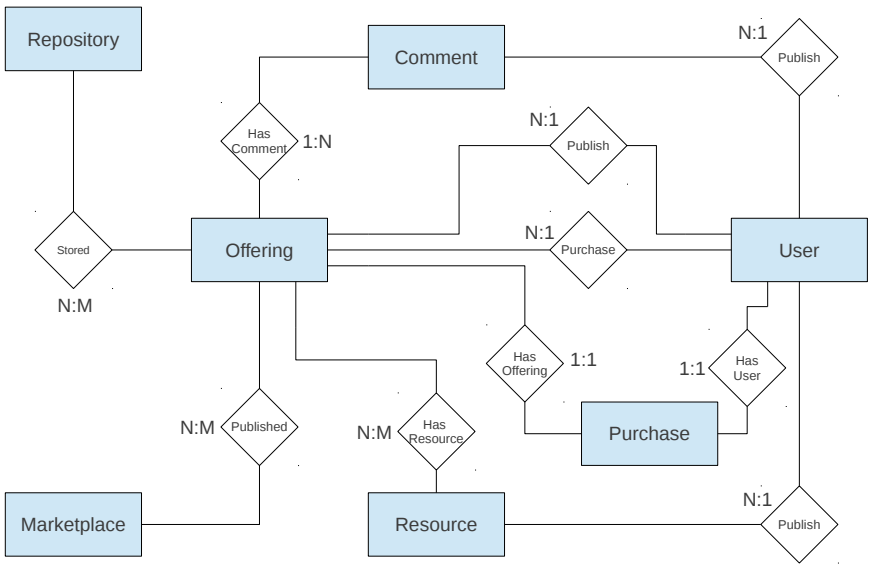

Figura 3.2: Modelo de datos del Store GE

En la Figura 3.2 puede verse de manera general el modelo de datos definido en el *Store GE* como un diagrama Entidad-Relación conteniendo las principales entidades en las que se basa el *Store GE* y las relaciones existentes entre ellas. A continuación se definen de manera detallada en que consisten estas entidades y que información debe recogerse de cada una de ellas.

Offering: Este modelo contiene la información relativa a las ofertas registradas en el *Store GE*.

- Name: Este atributo representará el nombre dado a la oferta por el proveedor.
- Version: Este atributo contendrá la versión de la oferta.
- Owner organization: Este atributo contendrá el nombre de la organización a la que pertenece la oferta.
- State: Este atributo representará el estado en el que se encuentra la oferta dependiendo de su ciclo de vida desde el punto de vista del proveedor.
- Model URL: Este atributo contendrá el enlace hacia la descripción de la oferta en *Linked USDL* [19] almacenada en una instancia del *Repository GE*.
- Rating: Este atributo contendrá el valor medio asignado por los distintos usuarios que han opinado sobre la oferta.
- TAGs: Este atributo contendrá una serie de *TAGs* usados para clasificar la oferta.
- Image: Este atributo contendrá la imagen asociada a la oferta.
- Related images: Este atributo contendrá una serie de imagenes relativas al servicio como capturas de pantalla, diagramas etc.
- Linked USDL information: Este atributo contrendrá la misma información que el documento *Linked USDL* almacenado en una instancia del *Repository GE* y será usado para poder tener acceso a la información de la oferta y al modelo de negocio sin necesidad de contactar constanteente con la instancia del *Respository GE*.

Es importante resaltar que en el modelo *Offering* la combinación de los atributos *Owner Organization*, *Name* y *Version* debe ser única ya que estos tres atributos serán usados para identificar la oferta, permitiendo de esta manera que dos ofertas pertenecientes a dos organizaciones distintas tengan distinto nombre y que puedan existir dos versiones diferentes de una misma oferta dentro de la misma organización. Nótese que el nombre del proveedor del servicio no es usado para identificar la oferta ya que se considera que es la organización la que realiza las ofertas.

Resource: Este modelo contiene la información relativa a los distintos recursos que han sido registrados en el *Store GE* para ser vinculados a alguna oferta.

- Name: Este atrbuto representará el nombre dado al recurso por el proveedor del mismo.
- Version: Este atributo representará la versión del recurso.
- State: Este atributo contendrá el estado del recurso indicando si este recurso está activo. Este atributo será utilizado principalmente en el caso de que el proveedor del recurso decida eliminarlo y éste ya forme parte de alguna oferta

publicada o incluso ya adquirida por algún comprador. En ese caso el recurso no puede ser eliminado ya que deberá seguir siendo accesible y por tanto simplente se marcará como borrado.

- Type: Este atributo indicará el tipo de recurso. Este tipo indicará si se trata de un recurso descargable como una aplicación o un widget, o un recurso asociado con el acceso a una API.
- Download link/Endpoint: Este atributo contendrá la información necesaria para el acceso al recurso. En el caso de que el recurso sea descargable contendrá el enlace para la descarga y en el caso en que se trate de una API contendrá el *Endpoint* de la misma.
- Description: Contendrá una descripción del recurso.

Es necesario indicar que los recursos en el *Store GE* serán identificados usando el nombre del usuario proveedor del recurso, el atributo *Name* y el atributo *Version* por lo que la combinación de estos tres elementos ha de ser única.

User: Este modelo contendrá información de los usuarios propia del *Store GE* que en principio no tiene por qué ser gestionada por el *Identity Management GE*

- User Name: Este atributo contendrá el nombre de usuario que identifica al usuario.
- Name: Este atributo contendrá el nombre completo del usuario.
- Organizations: Este atributo contendrá las organizaciones a las que pertenece el usuario.
- Roles: Este atributo contendrá los distintos roles que tiene el usuario.
- Default Tax Address: Este atributo contendrá la dirección fiscal por defecto proporcionada por el usuario.

Purchase: Este modelo contendrá información de las compras realizadas a través del *Store GE*

- Reference: Este atributo contendrá la referencia de la compra y será utilizado para identificarla.
- Date: Este atributo contendrá la fecha en la que se realizó la compra.
- State: Este atributo contendrá el estado de la compra, indicando si se ha realizado el pago o no.
- Bill: Este atributo contendrá una referencia la factura generada al realizar la compra.

- Tax Address: Este atributo contendrá la dirección fiscal efectiva usada por el usuario comprador de la oferta.

Marketplace: Este modelo contendrá información de aquellas instancias de *Marketpaces GE* en los que la instacia del *Store GE* ha sido registrada, con el objetivo de permitir a los proveedores elegir aquellos *Marketplaces GE* en los que su oferta será publicada.

- Name: Este atributo contendrá el nombre dado a la instancia del *Marketplace GE* por el administrador encargado haber registrado la instancia del *Store GE*.
- Host: Este atributo contendrá el *Endpoint* de la instancia del *Marketplace GE* en el que ha sido registrada la instancia del *Store GE*.

Repository: Este modelo contendrá información de aquellas instancias del *Repository GE* que han sido registradas en la instancia del *Store GE* con el objetivo de permitir elegir en que instancias del *Repository GE* se quieren almacenar los documentos *Linked USDL* que describen el servicio.

- Name: Este atributo contendrá el nombre dado a la instancia del *Repository GE* por el administrador encargado de haber registrado esta instancia..
- Host: Este atributo contendrá el *Endpoint* de la instancia del *Repository GE* registrada en la instancia del *Store GE*.

Comment: Este modelo contendrá información de aquellos comentarios realizados por los usuarios acerca de las distintas ofertas.

- Comment: Este atributo contendrá el comentario realizado por el usuario acerca de la oferta.
- Date: Este atributo contendrá la fecha en la que se realizó el comentario.
- Rating value: Este atributo contendrá el valor asignado a la oferta por el usuario.

3.1.5. Ciclo de vida de una oferta

Para especificar los distintos estados por los que puede pasar una oferta dentro del *Store GE* es necesario mostrar el ciclo de vida de una oferta desde dos puntos de vista diferentes. Por un lado el ciclo de vida de una oferta desde el punto de vista del proveedor de la misma (Fig. 3.3) y por otro el ciclo de vida de una oferta desde el punto de vista de un comprador (Fig. 3.4).

En la Figura 3.3 pueden verse los distintos estados por los que puede pasar una oferta desde el punto de vista de un proveedor. En primer lugar el proveedor del servicio crea la oferta, que estará en estado *Uploaded*. En este estado la oferta aún no está a la venta, sino que sólo es visible por el usuario que la ha creado , de esta manera

es posible que el proveedor registre distintos recursos en el *Store GE* sin necesidad de modicar una oferta que pudiera haber sido comprada. En el estado *Uploaded* la información asociada a la oferta es editable, pudiendo modificarse el documento Linked USDL [19] que define la información de la misma. Una vez se vinculan recursos a una oferta, esta pasa al estado *Bounded* y se mantendrá en este estado mientras la oferta no esté a la venta y existan recursos vinculados con la oferta. En caso de que todos los recursos de la oferta sean desvinculados, está regresará al estado *Uploaded*. Cuando el proveedor decide poner un servicio a la venta éste pasa al estado *Published*. Es al pasar a este estado cuando el *Store GE* registra la oferta en los *Marketplaces GE* indicados por el proveedor. Además, una vez una oferta ha pasado al estado *Published* ésta deja de ser editable, por lo que ya no es posible modificar la información del documento *Linked USDL* [19] ni vincular o desvincular recursos. Finalmente el proveedor del servicio puede borrar la oferta en cualquier momento. No obstante esta acción tendrá consecuencias diferentes dependendo del estado en el que se encuntre la oferta. Si la oferta aún no se había puesto a la venta, es decir, se encontraba en el estado *Uploaded* o el estado *Bound*, la oferta simplemente se elimina del *Store GE* ya que era algo local al proveedor. En el caso de que la oferta se encuntre en el estado *Published* está pasa al estado *Deleted* y deja de ser visible para futuros compradores. De esta manera aquellos compradores que ya hubieran adquirido la oferta siguen teniendo acceso a los recursos de la misma.

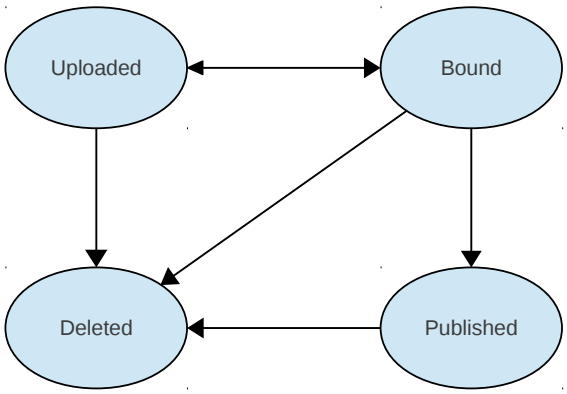

Figura 3.3: Ciclo de vida de una oferta desde la perspectiva del proveedor

En la Figura 3.4 pueden verse los distintos estados por los que puede pasar una oferta desde el punto de vista de un comprador. Desde la perspectiva de un comprador el primer estado que puede tener una oferta es el de *Published*. En este estado el comprador es capaz de ver toda la información asociada a la oferta que le

permita decidir adquirirla. Una vez el comprador se ha decido a adquirir la oferta ésta pasa al estado *Purchased* indicando que el comprador ha adquirido la oferta. En el caso de que el servicio adquirido necesite algún tipo de configuracíon previa al uso del mismo, el comprador podrá decidir el momento de realizar la configuracón, lo que supondrá el paso de la oferta al estado *Configurable*. En caso de que el servicio no requiera configuración la oferta pasará directamente al estado *Accessible*. De igual manera la oferta también pasará a este estado una vez realizada la configuración. En el estado *Accesible* ya se ha terminado el proceso de compra y los recursos asociados con la oferta son accesibles o descargables por el consumidor, dependiendo del tipo de servicio que se esté ofreciendo. Finalmente, en el caso de que que el consuminor ya no esté interesado en seguir teniendo el servicio, por ejemplo en el caso de un servicio de subscripción al que no quiere seguir subscrito, éste regresará al estado *Published* siendo posible para un consumidor volver a adquirir el servicio.

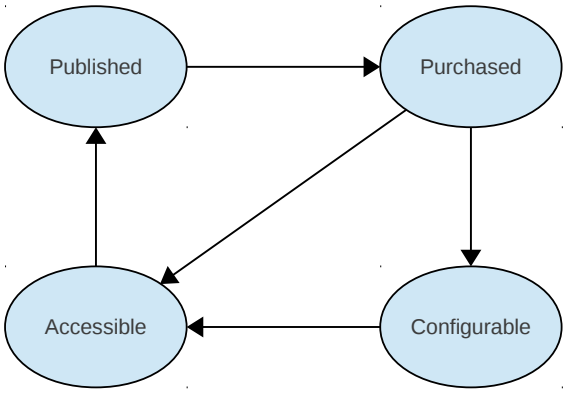

Figura 3.4: Ciclo de vida de una oferta desde la perspectiva del comprador

3.1.6. Casos de uso

En esta sección se describe el funcionamiento del *Store GE* desde el punto de vista de los usuarios, sin tener en cuenta las interacciones que se producen entre distintos *Generic Enablers*. Para ello se proponen algunos casos concretos de uso que hacen referencia a futuras interacciones reales de usuarios del *Store GE* tanto desde su portal Web como desde otros *Generic Enablers* que hacen uso de su API RESTful.

Creación y publicación de una oferta

En este caso de uso se describe el proceso que seguiría un proveedor de un servicio, desde el momento en que decide poner a la venta un nuevo servicio hasta que éste queda públicado en una instancia del *Marketplace GE*. Para ello se supondrá que el usuario con el rol *Provider* ha creado un nuevo *Mashup* haciendo uso del *Application Mashup GE*, por lo que en este caso la oferta a publicar tendrá un recurso descargable con el código de los distintos *Widgets* utilizados. Además la interacción con el *Store GE* se realizará desde la interfaz proporcionada por el *Application Mashup GE*.

En primer lugar el usuario compondrá el *Mashup* haciendo uso de las herramientas proporcionadas por el *Application Mashup GE*. Una vez terminada esta composición, cuando el usuario decida poner a la venta su servicio, la primera tarea que realizará sera crear un documento *Linked USDL* en el que se describirá toda la información relevante del servicio que va a poner a la venta, incluyendo la información básica de servicio (Nombre, fecha de última modificación, etc.), la información de tarificación, los términos y condiciones de uso y la información de nivel de servicio. Con esta descripción ya creada, el usuario podrá crear la oferta incluyendo algunas imágenes relativas a la misma como diagramas o capturas de pantalla, así como un icono para la misma. Es necesario destacar que en este punto la oferta ya esta creada pero aún no esta a la venta y sólo es visible por el usuario que la ha creado. El siguiente paso consistirá en el registro del recurso creado (Código del *Mashup*) dentro del *Store GE*, durante este paso el proveedor podrá ofrecer una URL desde la que sea posible descargar este recurso o podrá subir el recurso directamente al *Store GE* indicando que se trata de un recurso descargable. Con la oferta creada y el recurso registrado el siguiente paso dado por el usuario consitirá en la vinculación entre la oferta y el recurso, indicando de esta manera el código concreto que está siendo ofrecido. Finalmente el usuario pondrá a la venta su oferta indicando en que instancias del *Marketplace GE* quiere que su oferta sea publicada.

Búsqueda y adquisición de una oferta

En este caso de uso se describe el proceso que seguiría un usuario con el rol *Customer* desde el momento en que busca ofertas a través del portal Web proporcionado por el *Store GE* hasta la adquisición del mismo. En este caso se supone que el usuario hace uso del portal Web del *Store GE* y que adquiere una oferta con recursos asociados a la API de un servicio Web con un modelo de subscripción. Además se supone que el usuario tiene permiso para adquirir ofertas para toda su organización.

El primer paso a dar por el usuario consitirá en realizar el *login* dentro de la plataforma FI-WARE. Esto identificará al usuario dentro del *Store GE* así como a la organización a la que pertenece. Una vez el usuario está autenticado éste accederá al portal Web del *Store GE*, donde seleccionará la opción de búsqueda avanzada. Esto le permitirá filtrar por el tipo de los recursos asociados a la oferta. Al realizar la

búsqueda, se le mostrarán una serie de ofertas que satisfacen los parámetros pedidos, entre las cuales el usuario seleccionará aquella que mejor se ajuste a sus necesidades y preferencias. Cuando el usuario haya decidido adquirir una oferta este seleccionará la opción de adquirir la oferta para toda su organización, trás lo cual el *Store GE* solicitará la información necesaria para realizar el pago (Número de cuenta, número de tarjeta, cuenta de Paypal, etc.). Una vez recibida la confirmación del pago, el *Store GE* descargará la factura de la transacción realizada y redireccionará al usuario hacia el proveedor del servicio para obtener las credenciales de acceso a éste último. Cuando este proceso haya terminado, esta oferta y sus recursos asociados pasarán a formar parte de las ofertas propias de todos los usuarios de la organización desde donde podrán descargar la factura u obtener las credenciales de acceso.

3.1.7. Interacciones principales

En esta sección se describen de manera detallada las principales interacciones entre los distintos *Generic Enablers* y el *Store GE* con el objetivo de mostrar cómo éste encaja dentro de la arquitectura existente de la plataforma FI-WARE y cuál es la funcionalidad principal ofrecida al resto de *Generic Enablers*.

Crear una oferta

En la Figura 3.5 pueden verse las interacciones realizadas para la creación de una nueva oferta dentro del *store GE*. En este diagrama se supone el *Application Mashup GE* como cliente.

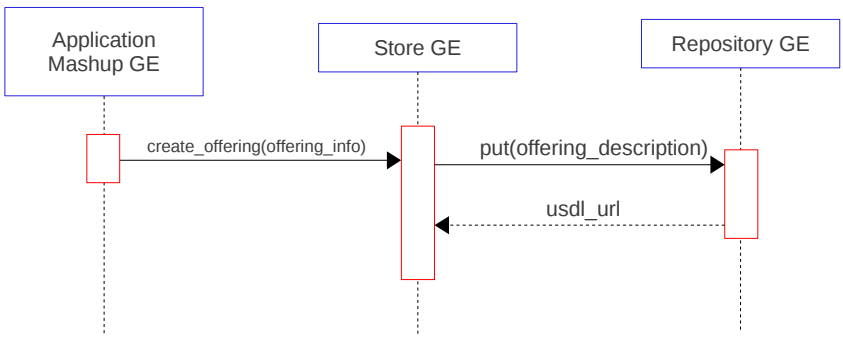

Figura 3.5: Creación de una oferta

Puede observarse en la Figura 3.5 cómo la creación de una nueva oferta está compuesta por una sola petición en la que se provee la información de la oferta. Como se comentó anteriormente esta información estará compuesta por la descripción *Linked USDL* [19] de la oferta, así como por una serie de imagenes relativas a la oferta y por el icono de la misma. Una vez que el *Store GE* ha recibido la petición para crear la nueva oferta éste realiza una petición al *Repository GE* para almacenar el documento *Linked USDL* [19] obteniendo una URL qie apunta al mismo y que será usada posteriormente para el acceso a esté documento y para el registro de la oferta en una instancia del *Marketplace GE*.

Añadir y vincular un recurso

En la Figura 3.6 pueden verse las distintas interacciones realizadas para el registro y vinculación de un nuevo recurso. En este diagrama se supone que el usuario está utilizando el portal Web del *Store GE*.

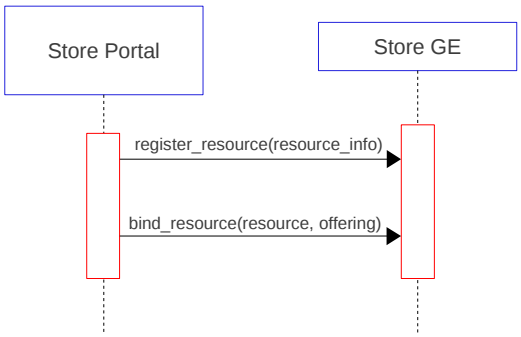

Figura 3.6: Registro y vinculación de un recurso

Puede observarse en la Figura 3.6 como el registro y vinculación de un recurso a una oferta son operaciones simples compuestas de una única petición. La petición de creación del recurso incluirá el nombre del recurso, la versión, una descripción, el tipo y la información necesaria para el acceso al recurso que podrá ser un *Endpoint* en el caso de que se trate de un recurso de acceso a una API o un enlace de descarga o el propio recurso en el caso de que se trate de un recurso descargable. Por otra parte, la petición para vincular el recurso a una oferta, incluirá en primer lugar la información para identificar el recurso, es decir, el nombre del proveedor, el nombre

del recurso y su versión, y por otra parte la información para identificar la oferta, es decir, la organización a la que pertenece, el nombre de la oferta y la versión de la misma.

Publicar una oferta

En la Figura 3.7 pueden verse las distintas interacciones realizadas para la publicación de una oferta creada en el *Store GE* en una instancia del *Marketplace GE*. En este diagrama se supone que el usuario hace uso del portal Web del *Store GE*.

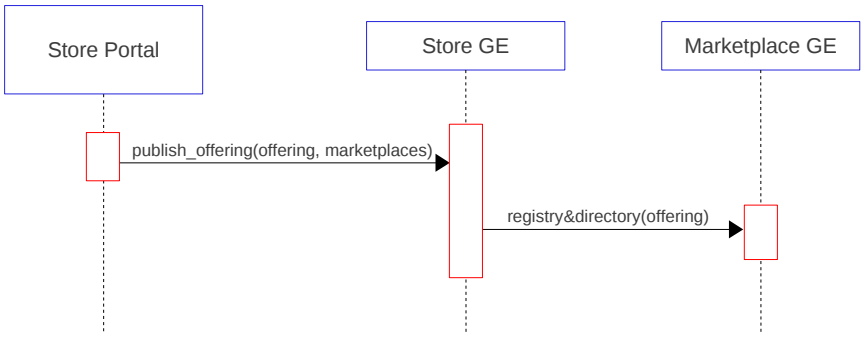

Figura 3.7: Publicación de una oferta

Se puede ver en la Figura 3.7 como se realiza la publicación de una oferta mediante una petición al *Store GE*. Esta petición contendrá en primer lugar la información necesaria para identificar la oferta, es decir, la organización a la que pertence, su nombre y su versión. Por otra parte contendrá además una lista de las intancias del *Marketplace GE* en los que se quiere publicar la oferta. Estas instancias estarán identificadas por el nombre que el administrador les dió cuando registró en ellas la instancia del *Store GE*. Cuando el *Store GE* recibe la petición de publicar la oferta contacta con el servicio *Registry & Directory* del *Marketplace GE*, que contendrá el nombre de la oferta y la URL de su descripción *Linked USDL* [19] almacenada en una instancia del *Repository GE*.

Eliminar una oferta

En la Figura 3.8 pueden verse las interacciones necesarias para la eliminación de una oferta creada en el *Store GE*, y publicada en algún *Marketplace GE*. Se supone

que el usuario hace uso del portal Web del *Store GE*.

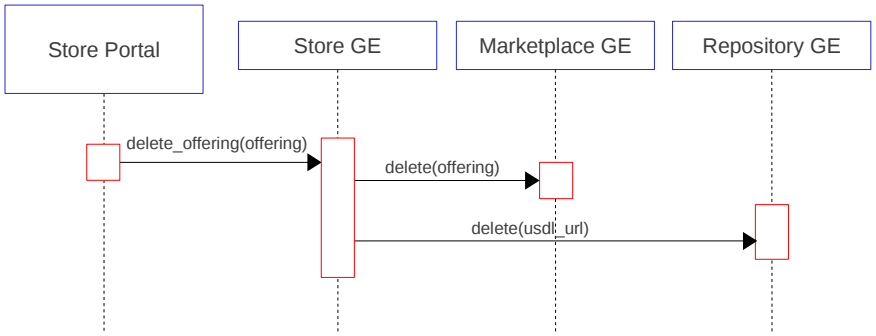

Figura 3.8: Borrado de una oferta

En la Figura 3.8 se puede ver como se realiza la eliminación de una oferta mediante una petición al *Store GE*. Esta petición contendrá la información necesaria para identificar la oferta, es decir, la organización a la que pertence, su nombre y su versión. Una vez el *Store GE* ha recibido la petición para eliminar la oferta, éste contactará en primer lugar con las intancias del *Marketplace GE* en las que la oferta esté publicada, indicando a estos que la oferta debe ser eliminada. En el caso del *Marketplace GE* las ofertas se identifican tan sólo por su nombre por lo que ésta será la única información contenida en dichas peticiones. Una vez eliminada la oferta de las intancias del *Marketplace GE*, el *Store GE* contactará con las distintas instancias del *Repository GE* usando la URL del documento *Linked USDL* [19] para eliminar está descripción.

Buscar en el Marketplace GE una oferta con recursos descargables y adquirirla

En la Figura 3.9 pueden verse las interacciones que se producirían al buscar ofertas en el *Marketplace GE* y al adquirir una de ellas, teniendo en cuenta que esta oferta estaría basada en una serie de recursos descargables registrados en el *Store GE*. En este diagrama se supone que la búsqueda y adquisición se realiza desde el *Application Mashup GE*.

En la Figura 3.9 se puede observar como el *Application Mashup GE* realiza una petición al servicio *Search & Discovery* del *Marketplace GE* conteniendo una serie

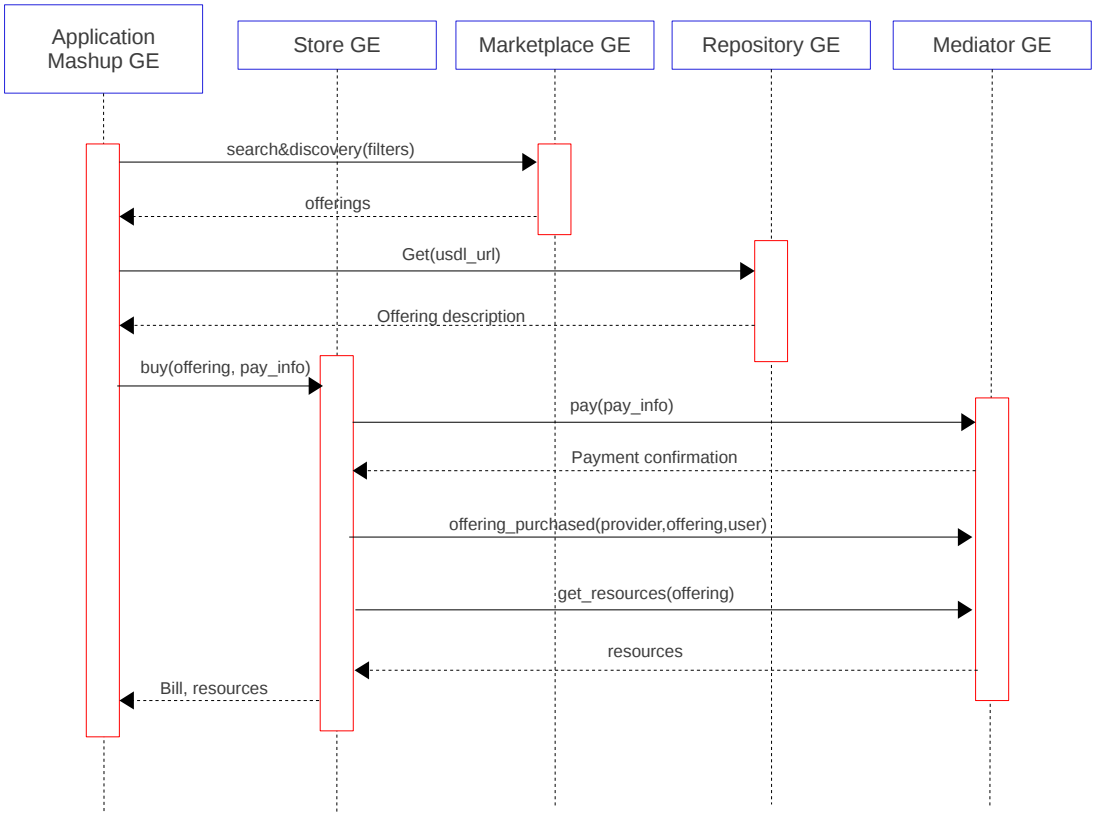

Figura 3.9: Búsqueda en el Marketplace GE y adquisición

de filtros que permitan acotar la búsqueda. El *Marketplace GE* contestará con la información de una serie de ofertas. Concretamente con el nombre de la oferta y la URL de las descripciones *Linked USDL*. Una vez recibido el resultado de la búsqueda, el *Application Mashup GE* usará esta información para descargar y visualizar las descripciones *Linked USDL* [19] de las ofertas, permitiendo al usuario la selección de una de ellas. Cuando se ha decidido la compra de una oferta, el *Application Mashup GE* realizará una petición al *Store GE* con la información que identifique la oferta: en este caso la URL de su descripción ya que es la información disponible en el *Marketplace GE* y la información para realizar el pago (número de cuenta, número de tarjeta, cuenta de paypal etc.). En el momento en que el *Store GE* recibe la petición de compra, contacta con la pasarela de pago correspondiente a través del *Mediator GE*, facilitando la información de pago. Cuando el *Store GE* recibe la confirmación del pago contacta con el proveedor del servicio para informarle de la compra y utiliza los enlaces de descarga de los recursos para obtenlos. Es necesario indicar que estas dos interacciones se realizarán a través del *Mediator GE*. Finalmente, el *Store GE* enviará al *Application Mashup GE* la factura de compra y los recursos asociados.

Buscar en el Store GE una oferta con recursos subscribibles y adquirirla

En la Figura 3.10 pueden verse la interacciones que se producirían al buscar ofertas en el *Store GE* y al adquirir una de ellas, teniendo en cuenta que esta oferta estaría basada en una serie de recursos de acceso a APIs. En este diagrama se supone que el usuario utilizaria el portal Web del *Store GE*.

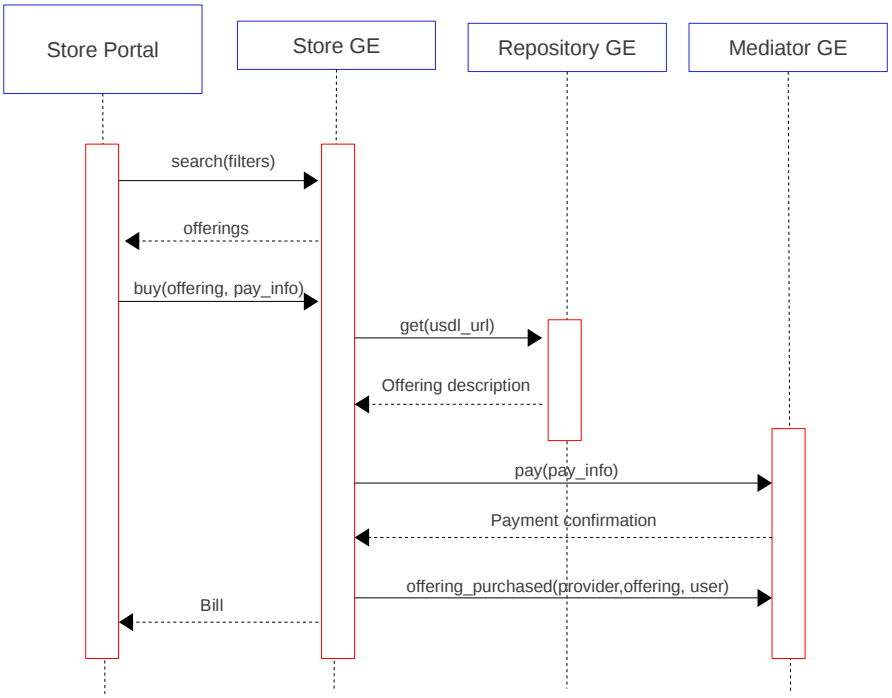

Figura 3.10: Búsqueda en el Store GE y adquisición

Se puede observar en la Figura 3.10 cómo se realiza una petición de búsqueda en el *Store GE*, pasando una serie de parámetros para acotar la búsqueda, a lo que el *Store GE* contestará con una lista de ofertas que satisfarán la búsqueda realizada. Cuando se ha decidido a adquirir una de esas ofertas, se realizará la petición de compra al *Store GE* incluyendo la información necesaria para identificar la oferta, es decir, la organización, el nombre y la version así como la información necesaria para realizar el pago (número de cuenta, número de tarjeta, cuenta de Paypal, etc.). Cuando el *Store GE* reciba la petición de compra contactará con el *Repository GE* utilizando la URL de la descripción de la oferta seleccionada para descargar el documento *Linked USDL* y comprobar la información sobre tarificación de la oferta. Seguidamente el

Store GE realizará el pago contactando con la pasarela de pago correspondiente a través del *Mediator GE*, usando la información de pago obtenida anteriormente. En cuanto el *Store GE* reciba la confirmación del pago contactará con el proveedor del servicio para indicarle que el usuario u organización ahora tiene acceso a las APIs definidas por los recursos de la oferta. Finalmente el *Store GE* enviará la factura de compra.

3.2. Implementación de referencia

En esta sección se define en que consiste la implementación de referencia realizada del *Store GE*. Puesto que el objetivo de esta implementación es realizar una prueba de concepto de la *Open Specification* definida anteriormente tan sólo se han implementado las partes imprescindibles para su realización, en concreto se ha implementado la integración con el *Marketplace GE*, la integración con el *Repository GE*, la funcionalidad necesaria para gestionar ofertas incluyendo la creación de ofertas, el registro de recursos, la vinculación de recursos a ofertas, la publicación de ofertas, y por último funcionalidad para la adquisición de ofertas.

Para la realización de esta prueba de concepto se han realizado una serie de simplificaciones sobre el modelo definido en la *Open Specification*. En primer lugar, puesto que aún no existe una implementación del *Identity Management GE* que realice la gestión compleja de las organizaciones, se considerará que un usuario pertenecerá a una sola organización que será representada simplemente como un nombre incluido en el modelo de usuarios de la base de datos, además se considerará que un usuario siempre que adquiere una oferta la adquirirá para toda su organización. Por otra parte, puesto que la prueba de concepto se realizará sobre la implementación de referencia del *Application Mashup GE* (plataforma *Wirecloud*) y que aún no existen servicios accesibles via API en la plataforma FI-WARE, se considerará que todos los recursos son descargables y serán almacenados dentro de la propia implementación del *Store GE*. Finalmente todas las ofertas se considerarán gratuitas ya que no tiene sentido incurrir en la complejidad legal y de seguridad que supondría manejar pagos reales para realizar una prueba de concepto.

Esta implementación de referencia está dividida en dos partes, por una parte el servidor de la aplicación, que contiene la funcionalidad asociada a las APIs y la base de datos, y por otra parte el portal Web que funcionará como cliente de la aplicación.

3.2.1. Servidor de la aplicación

El servidor de la aplicación está desarrollado utilizando el *framework* de desarrollo Django [6] en su versión *Django non-rel* que permite usar bases de datos no relacionales ya que se ha optado por la utilización de MongoDB [25] como base de datos de la aplicación.

Estructura de la base de datos

MongoDB es una base de datos no relacional que utiliza una estructura de documentos basados en JSON, por ello se ha adaptado el modelo de datos definido en la *Open Specification* para permitir representar las entidades y relaciones defindas como documentos dentro de una colección. A continuación puede verse la estructura básica de estos documentos.

A continuación se muestra la estructura básica de los documentos utilizados en la colección de ofertas, puede verse como se ha incluido un campo con el usuario proveedor del servicio, además las relaciones *Many-To-Many* del modelo de datos apacecen representadas como listas en el documento. Es necesario indicar que el campo offering_description hace referencia al atributo *Linked USDL Information* definido en el modelo de datos y que debe contener la información de la descripción del servicio, para realizar esto se ha optado por incluir la información del documento *Linked USDL* [19] tal cual en *RDF* [30] usando el formato de representación *JSON-LD* [7].

```
{
    "name": "Nombre de la oferta",
    "owner_organization": "Organizacion a la que pertenece",
    "owner_admin_user": "creador de la oferta",
    "version": "1.0"
    "state": "Uploaded"
    "description_url": "URL al Repository GE"
    "marketplaces": [],
    "resources": [],
    "rating": "5",
    "comments": [],
    "tags": [],
    "image_url": "URL al icono de la oferta",
    "related_images": [],
    "offering_description": {},

}
```

A continuación puede verse la estructura básica de los documentos pertenecientes a la colección de recursos. Es necesario indicar que el campo *download_link* contendrá un enlace para realizar la descarga del recurso desde el servidor de aplicaciones del proveedor en caso de que se haya elegido este método y que el campo *resource_path* contendrá la ruta al recurso en caso de que este haya sido guardado directamente en el *Store GE*. Finalmente, el campo *offerings* contendrá una lista de las ofertas en las que está vinculado, este campo representa la información asociada a la relación *Many-To-Many* existente entre *Offering* y *Resource*.

```
{
      "name": "Nombre del recurso",
      "version": "1.0",
      "provider": "Proveedor del recurso",
      "resource_type": "download",
      "description": "Descripcion del recurso",
      "state": "registered",
      "download_link":  "Enlace de descraga del recurso"
      "resource_path": "Ruta al recurso"
      "offerings": []
}
```

Puesto que Django [6] ya tiene un modelo de usuarios se ha definido una colección *User Profile* con la estructura de documentos que se encuentra a continuación para representar la información necesaria de los usuarios y se ha vinculado al modelo de usuarios de Django [6] haciendo uso del campo *user*.

```
{
      "user" : "Referencia al usuario",
      "organization": "Organizacion del usuario",
      "roles": [],
      "offerings_purchased": [],
      "offerings_provided": []
}
```

A continuación puede verse la estructura de documentos utilizada para representar la información de la entidad *Purchase* y sus relaciones. Puede verse que se hace uso de referencias para apuntar a los otros modelos, además el campo *bill* contendrá la ruta a la factura de compra en formato PDF.

```
{
    "ref": "N de referencia",
    "customer": "Referencia al comprador",
    "date": "Fecha de compra",
    "offering": "Referencia a la oferta"
    "state": "Paid"
    "bill": "Ruta a la factura",
    "tax_address": "Direccion fiscal de la compra"
}
```

Finalmente los documentos asocidados a los modelos *Marketplace* y *Repository* estarán compuestos por su nombre y su endpoint de la misma manera que en las modelos definidos en la *Open Specification* por lo que no se ha considerado necesario incluir su estructura en este documento.

Diseño e implementación

Para el desarrollo del servidor de la aplicación se han hecho uso de las facilidades proporcionadas por Django [6] para crear una arquitectura Modelo-Vista-Controlador de manera sencilla. Se han creado los modelos de la base de datos haciendo uso de los ficheros models.py y se han incluido las distintas APIs en los

ficheros urls.py. El servidor de la aplicación se organiza en torno a cuatro apps de Django, éstas son *store_commons*, *market_adaptor*, *repository_adaptor* y *fiware_store*. La app de Django [6] *store_commoms* contiene una serie de librerías y funciones generales usadas por los distintos componentes de la aplicación. Un ejemplo de las funcionalidades contenidas en esta app es la clase *MethodRequest* que permite realizar peticiones HTTP indicando el método HTTP utilizado. La app de Django [6] *market_adaptor* contiene la funcionalidad necesaria para las comunicaciones con el *Marketplace GE* y la app de Django [6] *repository_adaptor* se encarga de proveer de la funcionalidad necesaria para la comunicación con el *Repository GE*. Finalmente la app de Django [6] *fiware_store* contiene la funcionalidad principal del *Store GE* dividida en distintos módulos, tambíen es dentro de esta app donde se declaran los modelos de la base de datos y las URLs de las APIs.

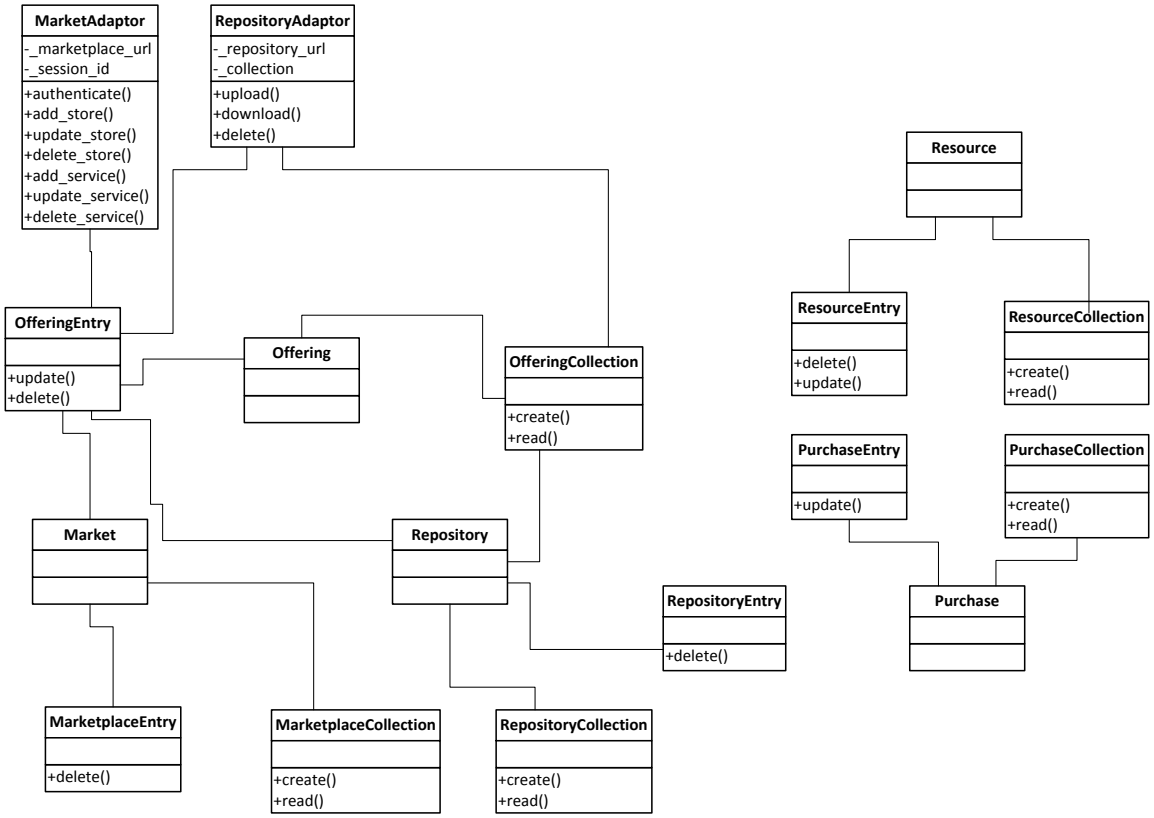

Figura 3.11: Esquema general del lado servidor

.

En la Figura 3.11 puede verse el esquema del servidor implementado teniendo en cuenta que las clases *Market*, *Repository*, *Offering*, *Resource* y *Purchase* hacen referencia a los distintos modelos de la base de datos ya que estos son gestionados en Django [6] como clases. Por otra parte las clases *MarketAdaptor* y *RepositoryAdaptor*

son las utilizadas para realizar la comunicación con el *Marketplace GE* y el *Repository GE* respectivamente y abstraen estas comunicaciones al resto de la aplicación. Finalmente es necesario indicar que el resto de clases existentes están vinculadas a los distintos *endpoints* de la API del servidor teniendo un método para cada uno de los métodos HTTP (create - POST, update - PUT, etc.).

3.2.2. Portal Web

El portal Web está realizado utilizando el leguaje de programación JavaScript [17] junto con los *templates* de Django [6] que contendrán el código HTML del portal, además se han utilizado hojas de estilo en cascada (CSS) y la librería JQuery [18].

Diseño e implementación

El diseño del portal Web gira en torno a tres vistas principales cada una ellas con su propio documento HTML (generado a partir de un *template* de Django [6]) que cargan una serie de ficheros JavaScript [17]. Estas tres vistas son *Administration*, *Catalogue* y *Store*.

La vista *Administration* está compuesta por un documento HTML general y por una serie de ficheros JavaScript encargados tanto de los cambios dinámicos en la interfaz como de proveer funcionalidad de administración, en concreto esta vista permite realizar tanto las tareas de registro de instancias del *Repository GE* como las de registro de la instancia del *Store GE* en diversos *Marketplace GE*. Es importante resaltar que esta vista sólo estará disponible si el usuario tiene el rol *Admin*.

En la Figura 3.12 puede verse el diagrama de clases perteneciente a la vista de administración. En este digrama se observa que la funcionalidad propia de tareas de administración se encuentra en una sola clase llamada *AdminView*. Ademas está incluida la clase *EndpointManager* encargada de abstraer las URL asociadas a las APIs del servidor y la clase *MessageManager* encargada de generar distintos tipos de ventanas de mensajes.

La funcionalidad provista por los ficheros JavaScript [17] asociados a la vista *Catalogue* se encarga de mostrar e interactuar con las distintas ofertas propias del usuario, es decir esta vista trabajará con las ofertas adquiridas por la organización del usuario, permitiendole descargar los recursos u obtener la factura de compra. Si el usuario tiene además el rol *Provider* esta vista permitirá visualizar sus ofertas creadas permitiendo eliminarlas, ponerlas a la venta, vincular recursos, etc. Además se encarga de la creación de las nuevas ofertas y del registro de nuevos recursos.

En la Figura 3.13 puede verse el diagrama de clases perteneciente a la vista *Catalogue*. En este diagrama pueden verse distintas clases asociadas con las distintas ventanas existentes en esta vista. En primer lugar la clase *CatalogueView* es la clase principal de esta vista y se encarga de pintar la ventana inicial. Por otra parte la

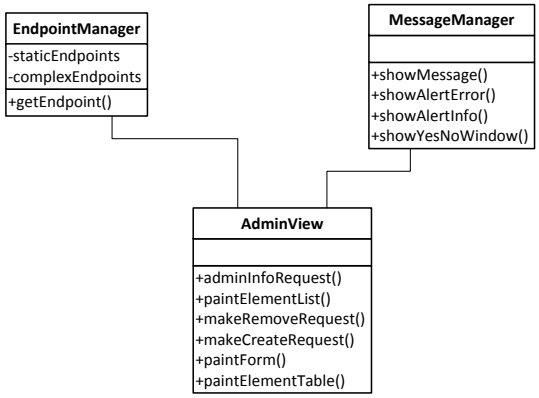

Figura 3.12: Diagrama de clases de la vista de administración

.

clase *CatalogueSearchView* se encarga de pintar las listas de ofertas del usuario tanto las adquiridas como las creadas. En siguiente lugar la clase *CatalogueDetailsView* se encarga de mostrar la infrrmación detallada asociada a una oferta concreata. Las clases *CreateOfferingForm* y *RegisterResForm* se encargan de crear los formularios para la creación y registro de ofertas y recursos respectivamente, además serán estas clases las que realicen estas peticiones al servidor. La clase *BindResources* en la encargada de mostrar y de permitir vincular los recursos exitentes con la oferta que se estuviera mostrando en ese momento. Finalmente la clase *PublishForm* creará el formulario para publicar una oferta y realizará la petición al servidor. Es necesario destacar que las distintas ofertas quedan representadas en esta vista como instancias de la clase *OfferingElement*. Por otra parte hay que destacar también que las distintas clases incluidas en el diagrama hacen uso de las clases *EndpointManager* y *MessageManager* definidas anteriormente y que no han sido incluidas por claridad del diagrama.

Finalmente en la vista *Store* se proporcionará la funcionalidad para realizar y visualizar búsquedas así como la funcionalidad necesaria para realizar la adquisición de las mismas.

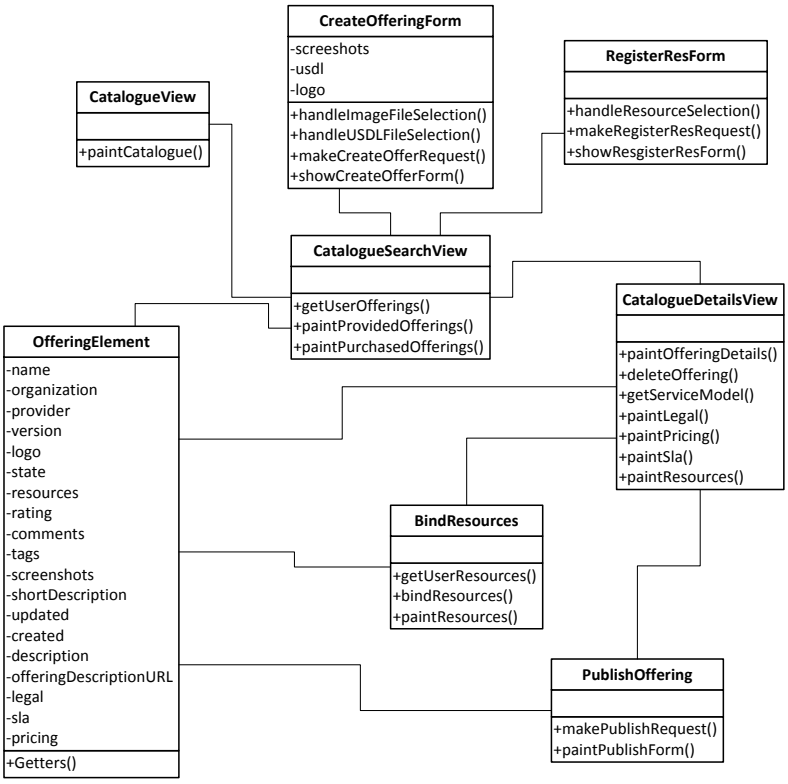

Figura 3.13: Diagrama de clases de la vista del catálogo

.

Capítulo 4

Conclusiones

4.1. Prueba de concepto

Como ya se ha comentado con anterioridad uno de los objetivos principales de este proyecto era la realización de una prueba de concepto, usando la primera versión de la implementación de referencia realizada también en al mismo, para comprobar el nuevo funcionamiento de la arquitectura del *Applications/Services Ecosystem* al incluir una instancia de esta implementación. A partir de la realización de esta prueba de concepto se han extraído una serie de conclusiones que se exponen en esta sección.

Para realizar esta prueba de concepto se ha hecho uso del portal Web del *Store GE*, de la implementación de referencia del *Application Mashup GE* (la plataforma *Wirecloud* [26]) y de las implementaciones de referencia del *Marketplace GE* y del *Repository GE* desarrolladas por SAP.

Esta prueba se organiza en torno a varias fases realizas por usuarios, con distintos roles y haciendo uso de las distintas interfaces y APIs ofrecidas por las implementaciones antes mencionadas, y ha supuesto la modificación del funcionamiento de la arquitectura comentado en el apartado 1.1.3. Las distintas fases de esta prueba se definen a continuación:

1. Un usuario con el rol *Admin* hace uso del portal Web de la implementación del *Store GE* para registrar la instancia del *Repository GE* y para registrar la instancia del *Store GE* en una instancia del *Marketplace GE*. En este punto las distintas conexiones entre las implementaciones ya están realizadas. Nótese que no es necesario ningún tipo de registro del *Store GE* en la plataforma *Wirecloud* ya que ésta tiene registrada con anterioridad la instancia del *Marketplace GE* que, a su vez, tiene la información del *endpoint* de la instancia del *Store GE*.

2. Un usuario con el rol *Provider* crea una nueva oferta haciendo uso del portal Web del *Store GE* pasando la información necesaria.

3. El usuario anterior registra un nuevo recurso haciendo uso del portal Web del *Store GE*. En este caso el recurso debe poder ser utilizado realmente en la

plataforma *Wirecloud*, por lo que se tratará del código de un *Widget* comprimido en un paquete siguiendo el formato propio de esta plataforma.

4. El usuario anterior vinculará la oferta creada con el recurso registrado y publicará esta oferta en la instancia del *Marketplace GE*.

5. Un usuario con el rol *Customer* realiza una búsqueda en la instancia del *Marketplace GE* haciendo uso de la interfaz de usuario desarrollada en la plataforma *Wirecloud*.

6. El usuario anterior realiza la adquisición de la oferta anteriormente publicada desde la plataforma *Wirecloud*, lo que supone la descarga del *Widget* empaquetado, que podrá ser incluido en la plataforma.

Hay que realizar algunas consideraciones técnicas con respecto a esta prueba de concepto. En primer lugar la creación de la nueva oferta se realiza desde el portal Web del *Store GE* debido a que la plataforma *Wirecloud* aún no soporta el ciclo de vida de una oferta desde el punto de vista del proveedor, aunque si es capaz de realizar la compra. Por otra parte se pretende que en el futuro la autenticación en todas las instancias de *Generic Enablers* sea única y realizada por el *Identity Management GE*. No obstante esta funcionalidad aún no está disponible, por lo que en aquellas interacciones en las que el usuario utiliza tanto la plataforma *Wirecloud* como el *Store GE* será necesario que el usuario esté previamente autenticado en ambos sistemas.

Una vez realizada esta prueba de concepto, es necesario comentar las principales ventajas que ofrece el uso del *Store GE* respecto al funcionamiento descrito en la sección 1.1.3. En primer lugar el funcionamiento anterior al *Store GE* requería que el documento *Linked USDL* [19] tuviera ciertos campos de acceso a la descripción técnica del *Widget* o *Mashup* que debía estar accesible en una URL concreta para poder acceder a la misma. Esta aproximación ofrecía muy poca flexibilidad y sólo permitía el uso de la arquitectura para trabajar con *Widgets* y *Mashups* de la plataforma FI-WARE. Con la introducción del *Store GE*, el documento *Linked USDL* [19] puede tener cualquier estructura válida ya que es el *Store GE* el que se encarga de la vínculación de los recursos, por lo que se podrá tener una mayor capacidad expresiva a la hora de describir una oferta y se podrán tener ofertas más complejas en las que, por ejemplo, se oferte más de un recurso.

Por otra parte la aproximación descrita en el apartado 1.1.3 no permite de ninguna manera un modelo de negocio en lo que se refiere a que los distintos recursos de la plataforma *Wirecloud* sean directamente accesibles. Al introducir el *Store GE* se fuerza a usar la API de compras del mismo lo cual permite controlar el acceso a los recursos de una oferta. De esta manera es posible controlar qué usuarios han adquirido qué ofertas de manera global a la plataforma FI-WARE y no solo en la plataforma *Wirecloud*. Además será posible realizar en el futuro un proceso de compra real.

Por todo lo descrito anteriormente se considera que, a pesar de que la introducción del *Store GE* fuerza a realizar un mayor número de operaciones para adquirir una oferta, mejora de manera significativa las capacidades de la plataforma FI-WARE, haciendola más útil de cara a proveedores de servicios y usuarios finales.

4.2. Conclusiones personales

Como valoración personal del trabajo realizado en este proyecto me gustaría destacar lo positivo que ha sido de cara a mejorar las posibilidades de mi futuro laboral, ya que he adquirido conocimientos y experiencia trabajando en un proyecto real y de ámbito internacional con números*os Partners*, cada uno con sus propias necesidades, siguiendo unos plazos, realizando reuniones, documentación, entregables, etc.

Debo destacar también los conocimientos que he adquirido en distintas tecnologías que están siendo valoradas últimamente como HTML5, CSS3, JavaScript [17], Python [29] o algunos *frameworks* de desarrollo como Django [6], y metodologías y buenas prácticas, como la metodología *Agile* usada en el proyecto FI-WARE o el uso de sistemas de control de versiones como GIT [12].

Finalmente debo destacar la experiencia de mi paso por el CoNWeT Lab (*Computer Networks and Web Tecnologies Lab* [5]) donde he trabajado tanto en el proyecto asociado a este Trabajo de fin de grado como en la plataforma *Wirecloud*, ya que existe un ambiente agradable que anima a seguir trabajando.

4.3. Líneas futuras

Dentro del contexto de este Trabajo de fin de grado se ha incluido la primera fase del trabajo a realizar para el proyecto FI-WAREen relación con la provisión de una eStore complemente funcional, lo que ha incluido la definición de la *Open Specification* y una primera versión limitada de la implementación de referencia, usada para realizar una prueba de concepto de la especificación anterior. Por tanto, el trabajo que se realizará a partir de ahora consistirá en una evolución progresiva de la implementación de referencia hacia una versión definitiva y estable siguiendo la metodología *agile* propia del proyecto FI-WARE. Más concretamente existe la siguiente planificación:

- En Marzo de 2013 tendrá lugar el siguiente *SPRINT* del proyecto para el cual se pide la funcionalidad descrita por algunas de las denominadas *features*, más concretamente tendrá que haber sido realizada la funcionalidad descrita a continuación:

    - *FIWARE.FEATURE.Apps.Store.IntegrationWithTheMarketplaceAndRepository*, esta *feature* indica que el *Store GE* debe estar perfectamente integrado

con el *Marketplace GE* y con el *Repository GE*. Esta es una funcionalidad básica y vital del *Store GE* que ya ha sido implementada dentro del contexto del Trabajo de fin de grado.

- *FIWARE.FEATURE.Apps.Store.SupportForBasicManagementOfOfferings*, esta *feature* indica que el *Store GE* debe ser capaz de gestionar ofertas. Esto incluye la creación de ofertas, el registro de recursos, la vinculación de recursos y ofertas y la capacidad de buscar y adquirir ofertas sin tener en cuenta la realización de un pago real por ellas. Puede verse que gran parte de esta funcionalidad ya ha sido implementada dentro del contexto de este Trabajo de fin de grado. No obstante esta funcionalidad será revisada y ampliada, sobre todo la funcionalidad relacionada con las búsquedas.

- En Junio de 2013 tendrá lugar la denominada *Second Major Release* para la cual se pide la funcionalidad descrita por las siguientes *features*:

  - *FIWARE.FEATURE.Apps.Store.ServiceConfiguration*, esta *feature* especifica que el *Store GE* debe ser capaz de permitir realizar la configuración de aquellos servicios que lo requieran una vez han sido adquiridos.
  - *FIWARE.FEATURE.Apps.Store.SupportForSocialFeatures*, esta *feature* especifica que el *Store GE* debe ofrecer una serie de características sociales que permitan mejorar la experiencia de usuario tales como un soporte para realizar comentarios y valoraciones sobre ofertas, un soporte para recomendaciones de ofertas basado en el histórico de compras del usuario o incluso la posibilidad de valorar a distintos proveedores.
  - *FIWARE.FEATURE.Apps.Store.ProvideAWebBasedGUI*, esta *feature* especifica que el *Store GE* debe ofrecer una interfaz de usuario basada en un portal Web desde el cual sea posible hacer uso de toda la funcionalidad ofrecidad por el sistema.

- Finalmente para el siguiente año del proyecto se plantea la realización de la funcionalidad asociada a la *feature* siguiente:

  - *FIWARE.FEATURE.Apps.Store.SupportForBuyingAService*, esta *feature* especifica que el *Store GE* debe ser capaz de gestionar la compra real de las ofertas publicadas haciendo uso de determinadas pasarelas de pago y teniendo en cuenta las distintas implicaciones de seguridad y legales que conlleva esta funcionalidad.

Además hay que tener en cuenta que el documento de la *Open Specification* podrá ser modificado a lo largo de la vida del proyecto al tratarse de un documento vivo dependiente de las necesidades del proyecto y de los distintos *Generic Enablers* existentes.

Apéndice A

Open Specifications

Las *Open Specifications* del *Store GE* son ducumentos vivos que van cambiando dependiendo de las necesidades de la plataforma FI-WARE. A continuación se encuentran los enlaces a la *Open Specification* y a la *API Open Specification* de *Store GE* publicadas en la Wiki del proyecto FI-WARE.

`https://forge.fi-ware.eu/plugins/mediawiki/wiki/fiware/index.php/FIWARE.OpenSpecification.Apps.Store`
`https://forge.fi-ware.eu/plugins/mediawiki/wiki/fiware/index.php/Store\_Open\_API\_Specification\_(PRELIMINARY)`

Bibliografía

[1] Application GE Open Specification. https://forge.fi-ware.eu/plugins/mediawiki/wiki/fiware/index.php/FIWARE.OpenSpecification.Apps.ApplicationMashup. Accessed: 13/01/2013.

[2] AWS Marketplace. https://aws.amazon.com/marketplace. Accessed: 13/01/2013.

[3] Cherokee Marketplace. http://cherokee-market.com. Accessed: 13/01/2013.

[4] Cherokee Project. http://www.cherokee-project.com. Accessed: 13/01/2013.

[5] Computer Networks and Web Tecnologies Lab. http://conwet.fi.upm.es. Accessed: 13/01/2013.

[6] Django Project. https://docs.djangoproject.com. Accessed: 13/01/2013.

[7] Formato JSON-LD. http://en.wikipedia.org/wiki/JSON-LD. Accessed: 13/01/2013.

[8] Formato N-Triples. http://en.wikipedia.org/wiki/N-Triples. Accessed: 13/01/2013.

[9] Formato N3. http://en.wikipedia.org/wiki/Notation_3. Accessed: 13/01/2013.

[10] Formato Turtle. http://en.wikipedia.org/wiki/Turtle_(syntax). Accessed: 13/01/2013.

[11] Future Internet Public-Private Partnership. http://www.fi-ppp.eu/. Accessed: 13/01/2013.

[12] GIT. http://en.wikipedia.org/wiki/Git_(software). Accessed: 13/01/2013.

[13] Google PLay. https://play.google.com/store. Accessed: 13/01/2013.

[14] Hubspot. http://marketplace.hubspot.com/. Accessed: 13/01/2013.

[15] Hubspot App Marketplace. `https://app.hubspot.com/market/front/list`. Accessed: 13/01/2013.

[16] Hubspot Services Marketplace. `https://services.hubspot.com/`. Accessed: 13/01/2013.

[17] JavaScript Wikipedia. `http://en.wikipedia.org/wiki/JavaScript`. Accessed: 13/01/2013.

[18] JQuery Wikipedia. `http://es.wikipedia.org/wiki/JQuery`. Accessed: 13/01/2013.

[19] Linked USDL. `http://linked-usdl.org/`. Accessed: 13/01/2013.

[20] Linked USDL-Core. `http://linked-usdl.org/ns/usdl-core`. Accessed: 13/01/2013.

[21] Linked USDL-Pricing. `http://linked-usdl.org/ns/usdl-price`. Accessed: 13/01/2013.

[22] Linked USDL-SLA. `http://linked-usdl.org/ns/usdl-sla`. Accessed: 13/01/2013.

[23] Maketplace GE Open Specification. `https://forge.fi-ware.eu/plugins/mediawiki/wiki/fiware/index.php/FIWARE.OpenSpecification.Apps.Marketplace`. Accessed: 13/01/2013.

[24] Mediator GE Open Specification. `https://forge.fi-ware.eu/plugins/mediawiki/wiki/fiware/index.php/FIWARE.OpenSpecification.Apps.Mediator`. Accessed: 13/01/2013.

[25] MongoDB. `http://www.mongodb.org/`. Accessed: 13/01/2013.

[26] Plataforma Wirecloud. `http://conwet.fi.upm.es/wirecloud`. Accessed: 13/01/2013.

[27] Programación basada en prototipos Wikipedia. `http://en.wikipedia.org/wiki/Prototype-based`. Accessed: 13/01/2013.

[28] Proyecto FI-WARE. `http://www.fi-ware.eu/`. Accessed: 13/01/2013.

[29] Pyhton Wikipedia. `http://es.wikipedia.org/wiki/Python`. Accessed: 13/01/2013.

[30] RDF W3C. `http://www.w3.org/TR/2004/REC-rdf-primer-20040210/`. Accessed: 13/01/2013.

[31] Registry GE Open Specification. `https://forge.fi-ware.eu/plugins/mediawiki/wiki/fiware/index.php/FIWARE.OpenSpecification.Apps.Registry`. Accessed: 13/01/2013.

[32] Repository GE Open Specification. `https://forge.fi-ware.eu/plugins/mediawiki/wiki/fiware/index.php/FIWARE.OpenSpecification.Apps.Repository`. Accessed: 13/01/2013.

[33] USDL. `http://www.internet-of-services.de/index.php?id=288\&L=0`. Accessed: 13/01/2013.

[34] VMWARE Solution exchange. `https://solutionexchange.vmware.com/store/`. Accessed: 13/01/2013.

Este documento esta firmado por

	**Firmante**	CN=tfgm.fi.upm.es, OU=CCFI, O=Facultad de Informatica - UPM, C=ES
	Fecha/Hora	Tue Feb 11 23:34:53 CET 2014
	Emisor del Certificado	EMAILADDRESS=camanager@fi.upm.es, CN=CA Facultad de Informatica, O=Facultad de Informatica - UPM, C=ES
	Numero de Serie	630
	Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)